Abstract of "Algorithms for Analyzing Human Genome Rearrangements," by Crystal L. Kahn, Ph.D., Brown University, May 2011.

The human genome exhibits a rich structure resulting from a long history of genomic changes, including single base-pair mutations and larger scale rearrangements such as inversions, deletions, translocations, and duplications. The number and order of the genomic changes that resulted in the present-day human genome is not known, but can sometimes be inferred by comparison to the genomes of other species. In particular, genome rearrangements are modeled as operations on signed strings of characters representing blocks of conserved sequences. Genome rearrangement distance measures quantify the similarity between two or more genome sequences by counting the minimum, or most likely, number of rearrangement operations needed to transform one sequence into another. The development of efficient algorithms for computing genome rearrangement distances has been instrumental both in computing phylogenies for sets of known genetic sequences (such as gene families or the whole genomes of present-day species) and in constructing ancestral genome sequences.

In this thesis, we develop algorithms to study recent genome rearrangements in human and cancer genomes. We introduce a novel measure, called *duplication distance*, to quantify the similarity between two genomic regions containing segmental duplications. We give an efficient algorithm to compute the duplication distance between a pair of signed strings and provide several generalizations of duplication distance that also measure inversions and deletions. We demonstrate the utility of the duplication distance measure in constructing the evolutionary history of segmental duplications in the human genome using both parsimony and likelihood techniques. Further, motivated by recent cancer genome sequencing studies, we present a new algorithm for the *block ordering problem* of inferring a whole genome sequence from a partial assembly by maximizing its similarity to another genome.

Algorithms for Analyzing Human Genome Rearrangements

by
Crystal L. Kahn
B. A., Amherst College, 2004
Sc. M., Brown University, 2008

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Program in Computer Science at Brown University.

Providence, Rhode Island
May 2011

This dissertation by Crystal L. Kahn is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____
Benjamin J. Raphael, Director

Recommended to the Graduate Council

Date _____ _____
Sorin Istrail, Reader

Date _____ _____
Bernard Moret, Reader
(École Polytechnique Fédérale de Lausanne)

Date _____ _____
Charles Lawrence, Reader
Applied Mathematics

Approved by the Graduate Council

Date _____ _____
Peter Weber
Dean of the Graduate School

Curriculum Vitae

Crystal Louise Kahn was born in Mansfield, Ohio on August 27, 1982, the best day of her older brother's life as yet. As the daughter of two political science professors fond of sabbaticals, she traveled a lot and at age eight moved to Fairfield, CT. She was valedictorian of Fairfield High School, a National Merit Scholar, and a Governor's Scholar of CT in 2000. From 2000 to 2004, she attended Amherst College in Amherst, MA where she earned a Bachelor of Arts in Computer Science, magna cum laude and with distinction. She was also awarded the Amherst College Computer Science Award and was inducted into the Phi Beta Kappa honor society in 2004. She spent the year of 2004 to 2005 at the University of Padova in Italy on a Fulbright Scholarship. In 2005, she began her graduate studies at Brown University in Providence, RI. She earned her Masters in 2008 while working toward a doctorate in Computer Science. While at Brown, she received a National Science Foundation Graduate Research Fellowship and an Amherst College Memorial Fund Fellowship.

She currently lives in Cambridge, MA with her fiance who denies that he was once also a computer scientist.

Acknowledgments

First, I would like to thank my advisor, Ben Raphael, for introducing me to computational biology and for helping me to discover interesting problems involving the design of algorithms. I would also like to thank the other members of my thesis committee. Sorin Istrail, the fearless leader of the Center for Computational Molecular Biology at Brown, has been a mentor and has introduced me to many luminaries in the field. Bernard Moret's research has influenced and inspired many contributions in this thesis. Chip Lawrence's insightful comments on my research were instructional and encouraged me to explore new directions. Of course, I am also indebted to my student coauthors, Shay Mozes and Borislav Hristov, for their contributions and energy.

Many members of the computer science community at Brown provided friendship and moral support (and sometimes even interesting conversation related to computer science). In particular, I'd like to thank Anna Ritz, Cora Borradaile, Claire Mathieu, Aparna Das, Suzanne Sindi, and Fabio Vandin.

I am grateful for financial support from the National Science Foundation, Amherst College, and Brown University.

Finally, I would like to acknowledge my family for their support, generosity, and patience. I cannot thank my parents (Bev and Bob), my brother (Adam), and my fiance (Mike Valentine) enough for letting me lean on them and for helping me to keep my eyes on the prize.

Contents

⋆Parts of this thesis have appeared before in [36, 37, 33, 34]. I thank my coauthors, Ben Raphael, Shay Mozes, and Borislav Hristov, for their permission to use portions of these papers in this thesis.

List of Figures

INTRODUCTION

Genome rearrangement events are large changes that occur in a genome sequence as the result of a chromosomal rupture. The human genome contains evidence of many rearrangements that have occurred over the course of evolution. We model a genome as a signed string on a multiset of *synteny blocks* that represent contiguous DNA sequences, corresponding to genes or other markers, that are conserved across genomes. Common types of sequence rearrangement events that have been studied are insertions (that augment a sequence by inserting a string of blocks in the middle of it), deletions (in which a substring of blocks is deleted from the middle of a sequence), reversals (in which the order of a contiguous substring of blocks is reversed), translocations (which are similar to cut-paste operations within a sequence), and duplication transpositions (which are similar to copy-paste operations within a sequence). Genome rearrangement distances count the minimum number of operations required to transform one genome into another. Sorting genomes by rearrangements is a related problem in which the transforming sequence of rearrangements between a pair of genomes is reconstructed. There has been much work in designing efficient algorithms for computing rearrangement distances and for sorting genomes by rearrangements. They have been used by evolutionary biologists to infer phylogenetic trees on genomes for different species, to construct putative ancestral genome sequences, and to assign orthologous genes in related species, among other applications. Genome rearrangements can also be used to model somatic mutations that occur within a single tissue, such as in a tumor mass.

1.1 Contributions and Organization

In this thesis, we present a number of techniques for analyzing rearrangements that occurred recently in the evolution of the human genome or that occur as the result of somatic mutations in cancer genomes. The study of genome rearrangements has resulted in a rich literature of algorithms for computing distance metrics between pairs or sets of genomes.

We contribute to this body of work by introducing new distance metrics between genomes that contain repeated elements, a traditionally confounding type of comparison. Computational biologists have used models of rearrangement, or rearrangement distances, to compute putative rearrangement histories for genomic sequences. For example, rearrangement distances have been used to characterize paralogous gene families within a genome, identify orthologous genes in different species, compare whole-genome sequences of different species, infer ancestral versions of present-day genomes, and study somatic mutations in cancer genomes. We also use our novel rearrangement distances to compute putative rearrangement histories for regions of the human genome containing duplicates using both parsimony and likelihood criteria. Finally, we discuss a problem in which a genome rearrangement distance is used to complete a partially assembled genome sequence by maximizing its similarity to another known genome sequence. We suggest a multi-genome generalization of this problem be used in efforts to sequence genomes from a sample of cancer cells.

This thesis is organized into four main parts. In the rest of this chapter, we discuss how our work relates to several relevant results on genome rearrangement distances and algorithms. In Chapter 2, we consider the problem of comparing genomes that contain duplicated regions. In Chapter 3, we present problem formulations for computing duplication histories for regions containing segmental duplications and compute putative histories for a set of segmental duplications in the human genome. Finally, in Chapter 4, we give an algorithm for inferring a whole-genome sequence from a set of partially assembled contigs by maximizing similarity to a known reference genome. We also formulate the problem of inferring the content of a mixture of genomes from a set of measured rearrangements, a problem motivated by cancer sequencing studies. We summarize our main results here:

Duplication Distance We present a novel genome rearrangement distance, called duplication distance, that counts the number of duplicate operations needed to construct a given target sequence by copying and pasting substrings of a fixed source sequence. We present several generalizations of this basic duplication model that allow also for certain types of reversal and deletion operations. We give efficient algorithms for computing duplication distance and its variants.

Most-Parsimonious Two-Step Duplication Tree We introduce an integer linear program formulation of the problem of computing a putative history for a set of genomic regions

containing duplicates that is based on a parsimony criterion. We use our problem formulation to compute a putative two-generation evolutionary history for a set of segmental duplications in the human genome using duplication distance as a measure of parsimony.

Maximum Parsimony Evolutionary History DAG We introduce an optimization problem for computing an evolutionary history for a set of genomic regions containing duplicates where a history is represented as a directed acyclic graph (DAG). We use our problem formulation to compute a putative evolutionary history DAG for a set of human segmental duplications using duplication distance as a measure of parsimony.

Maximum Likelihood Evolutionary History DAG We develop a probabilistic model of segmental duplication based on a partition function of the weighted ensemble of duplication scenarios. We give an efficient algorithm for computing the partition function of an ensemble of duplication scenarios. We introduce a likelihood analog of the Maximum Parsimony Evolutionary History DAG optimization problem using our probabilistic model to compute likelihood scores. We use our problem formulation to compute a putative evolutionary history DAG for a set of human segmental duplications, and we compare the likelihood and parsimony solutions.

Completing a Partially Assembled Genome Using a Rearrangement Distance We give a simple algorithm for the *block ordering problem* that constructs a genome assembly from a set of partially assembled contigs so as to maximize the similarity between the resulting genome and a known reference genome. We present a linear-time algorithm for the problem when the measure of similarity is defined as the double-cut-and-join (DCJ) distance between a pair of genomes. We further provide a proof that given a pair of genomes, the number of cycles in their breakpoint graph is equal to the number of cycles in their adjacency graph. Finally, we suggest the problem of computing a most-parsimonious set of k genomes that collectively exhibit a set of measured rearrangements, motivated by recent paired-end sequencing studies of cancer genomes.

1.2 Related Work

Although the work presented in this doctoral thesis falls into the broad category of algorithms for analyzing genome rearrangements, the techniques and models employed are varied and draw inspiration from many seminal works. Our work includes both results in the design of rearrangement models and algorithms for computing rearrangement distances and results in the computational analysis of genetic data from the human genome. Here we

place our work into context with other related research efforts by other members of the community.

1.2.1 Models of Genome Rearrangement

Genomes evolve via many types of mutations ranging in scale from single nucleotide mutations to large genome rearrangements. In this thesis, we consider several types of large-scale genome rearrangements that are caused by chromosomal ruptures. Computational models of these mutational processes allow researchers to derive similarity measures, called *rearrangement distances*, between genome sequences and to reconstruct evolutionary relationships between genomes. For example, considering substring reversals as the only type of mutation leads to the so-called reversal distance problem of finding the minimum number of reversals that transform one genome into another [58, 53]. Developing genome rearrangement models that are both biologically realistic *and* computationally tractable remains an active area of research.

Traditionally, computational biologists model a genome as a string on an alphabet of *synteny blocks* that may represent genes or other genomic sequences that are conserved (either across multiple loci in a single genome or across multiple genomes exhibiting orthologous copies of the same sequence). In the literature, a genome that contains some synteny block in duplicate is *ambiguous* and one without duplicates (i.e. a permutation) is *non-ambiguous*. The first results in the area of genome rearrangement distances dealt with distances between non-ambiguous genomes. An early breakthrough in the study of genome rearrangements is due to Hannenhalli and Pevzner [30] who introduced a polynomial-time algorithm for computing the reversal distance between signed permutations (where every integer has a +/- orientation) in which the only rearrangement event considered is a reversal. For example, given a signed string $X = +1\ +2\,+3\ +4$, a reversal of the substring, $+3\,+4$, yields $X' = +1\ +2\,-4\ -3$. This result was later improved in [10] and then again in [4]. (For discussion see cf. [50] and references therein).

Several elegant extensions of the reversal distance model have also been considered. For example, [23] extends the theory of [30] to compute the distance between a pair of genomes that may not necessarily contain the same set of blocks by allowing insertions and deletions of substrings. For instance, the reversal distance between $X_1 = +1\ +2\,+3\ +4$ and $X_2 = +5\ -3\,-2\ +6$ is undefined but [23] computes a reversal distance that also allows operations that delete the blocks that only appear in one of the input genomes (i.e. blocks 1, 4, 5, and

6). This was later improved by [41].

Ambiguous genomes present a particular challenge for genome rearrangement analysis and often make the underlying computational problems more difficult. For instance, computing reversal distance in signed genomes with duplicates is NP-hard [19].

There have been, however, several efficient solutions given to problems involving genomes with duplicates. For example, the genome halving problem has been solved. The input to the problem is a genome with exactly two copies of every character and the goal is to construct a minimum sequence of reversals that transforms the input genome into any *doubled genome* equal to the concatenation of two identical non-ambiguous genomes. This problem was first explored by [25] who gave a solution that was later discovered to be incomplete and was corrected in [1].

Another type of rearrangement that has been used to compare ambiguous genomes is the tandem duplication model in which an operation copies a substring of the genome and reinserts it into the genome right next to itself. For example, [18] presented the tandem-duplication random-loss model. In one operation, a substring of the genome is duplicated and inserted right next to itself and then exactly one copy of each of the newly duplicated integers is deleted. [18] gives exact, polynomial-time algorithms for special cost functions. In [11], the authors consider tandem duplications in the context of inferring a most parsimonious sequence of tandem duplication, gene loss, speciation, and reversal events that is consistent with a given gene tree and such that the total number of reversals is minimized. In [26], the authors consider the problem of constructing the duplication history (i.e. phylogenetic tree) for a set of tandemly repeated genes. In a duplication tree, each leaf of the tree corresponds to one of the present-day paralogous genes, and each internal node corresponds to the duplication of either a single gene or a set of adjacent genes. They give a simple method for determining whether a given rooted phylogeny is also a partially ordered duplication history (i.e. agrees with the order of the genes). They also give an exhaustive search method for finding the max parsimony duplication history. In [3], the authors consider tandem, alpha-satellite repeats in the human genome. They construct a probabilistic framework for evaluating the likelihood that a particular set of tandem repeats evolved by the physical process of unequal crossover. Finally, in [20], the authors give a polynomial-time approximation scheme (PTAS) for computing the optimal history (i.e. duplication tree) of tandem duplications for a given ambiguous genome where nodes may correspond to tandem duplications of contiguous substrings, and the cost on an edge is the

hamming distance between the two sequences at the endpoints.

There have also been many proposed models that compute the distance between a non-ambiguous ancestral genome and a present-day ambiguous genome. In these models, the sequence of transforming rearrangements must include operations that introduce arbitrary duplicates into the genome. However, efficient algorithms to compute these distances exactly are largely unknown.

For example, [24] gives a method for computing the minimum number of duplication transpositions and reversals needed to transform any non-ambiguous ancestor into a given present-day, ambiguous genome. The method is not efficient unless the present-day genome contains no more than two copies of any duplicate, and even in this case, the algorithm presented in [24] is flawed. (See Appendix A for a discussion.) In [43], the authors give an efficient approximation algorithm that computes the distance between the identity permutation and an arbitrary (possibly ambiguous) genome under reversals, deletions, and duplicating insertions. This work is extended by [56] who compute the distance between two arbitrary (possibly ambiguous) genomes approximately. Unfortunately, no exact solution for this problem is known. An exact algorithm to compute a minimum distance between an arbitrary (ambiguous) genome and some non-ambiguous ancestral genome under duplication transpositions is given in [60], but the duplication transposition model relies on the simplifying no-breakpoint-reuse assumption allowing a simple greedy method to suffice.

In Chapter 2, we discuss a novel rearrangement distance, called *duplication distance*, introduced in [36], that models the duplication and transposition of contiguous genomic substrings *en bloc* between disparate loci. The duplication distance from a source string $\mathbf{x}$ to a target string $\mathbf{y}$ is the minimum number of substrings of $\mathbf{x}$ that can be sequentially copied from $\mathbf{x}$ and pasted into an initially empty string in order to construct $\mathbf{y}$. We derive an efficient exact algorithm for computing the duplication distance between a pair of strings. Note that the string $\mathbf{x}$ does *not* change during the sequence of duplication events. Moreover, duplication distance does not model local rearrangements, like tandem duplications, deletions or inversions, that occur within a duplication block during its construction. While such local rearrangements undoubtedly occur in genome evolution, the duplication distance model focuses on identifying the duplicate operations that account for the construction of repeated patterns within duplication blocks by aggregating substrings of other duplication blocks from different loci. Thus, like nearly every other genome rearrangement model, the duplication distance model makes some simplifying assumptions about

the underlying biology to achieve computational tractability. In [33, 35], we extended the duplication distance measure to include certain types of deletions and inversions, and we give polynomial-time exact algorithms for computing these extensions. The reversals we consider only occur within a particular duplicated segment of the source string before being inserted into the target; we do not allow arbitrary reversals to occur in the target string. The deletions we consider, however, are arbitrary substring deletions that can occur in the target string at any time during a sequence of operations. These extensions make our model less restrictive – although we still maintain the restriction that **x** does not change during the sequence of duplications – and allows the construction of more rich, and perhaps more biologically plausible, duplication scenarios. While not explicitly modeling every type of rearrangement that might occur within a sequence of operations that builds a target string, duplication distance (and its extensions) provide an approximation of how a sequence of operations might occur and is efficiently computable in polynomial time.

Moreover, the abstraction we make by distinguishing the fixed source string from the changing target string is inspired by a known biological process by which mosaic patterns of segmental duplications are composed within mammalian genomes. Thus, the source and target strings represent two distinct genomic regions that might possibly be on different chromosomes. This process, known as the *two-step model* of segmental duplication is discussed in greater detail in Chapter 3.

1.2.2 Multiple Genome Rearrangement Algorithms

Computing rearrangement histories for a set of more than two genomes is used in constructing phylogenies of species and identifying orthologous genes in different species among other applications. The simplest problem to define on a set of multiple genomes is the median problem with respect to a certain rearrangement distance. Given a set $\{G_1, G_2, \ldots, G_k\}$ of genomes, the median problem is to find a genome H that minimizes $\sum_{i=1}^{k} d(G_i, H)$ where d is some distance measure. The median problem has been shown to be NP-hard with respect to reversal distance on signed permutations [17] and with respect to the simpler breakpoint distance on both signed and unsigned permutations [49] where the breakpoint distance between a pair of genomes is the number of character adjacencies that are exhibited in one genome and not the other. A heuristic for computing a median permutation with respect to breakpoint distance has been given by [55] and an approximation algorithm for computing the median problem with respect to a special case of the tandem-duplication random-loss distance was given by [18].

The problem of constructing a phylogenetic tree to represent a rearrangement history for a set of known genomes of common ancestry has been well-studied. In the phylogenetic tree problem, the leaves of the tree are the set of known genomes. For example, [12] describes a heuristic (BPAnalysis) for computing the unknown ancestral genomes in a fixed phylogenetic tree with a breakpoint distance criterion. The method is exponential in both the number of genomes and the size of the genomes. In [22], the authors improve the method presented in [12]; their method is only exponential in the number of genomes. This was improved further by [46] in a tool called GRAPPA that also computed phylogenies with respect to reversal distance. This was then refined by [47] who also increased the speed by a factor of one million. In [16], the authors introduce a new heuristic (MGR) for computing phylogenies with respect to reversal distance that is shown to perform better than the method given in [47] on real data.

1.2.3 Analysis of Duplicated Genomic Regions

Computational biologists use genome rearrangement distances to infer evolutionary relationships between species or to infer the history of genomic regions of interest. Many computational biologists are interested in reconstructing the histories of regions containing duplicated segments. For example, in [8], the authors construct a phylogeny for a set of species using regions containing orthologous repeats under the "no homoplasy" assumption.

The Alu family of repetitive elements has been studied in detail as certain Alu insertions or mutations have been linked to several human diseases. In [45], the authors do a limited Alu phylogeny reconstruction by recursively computing a maximum likelihood partition of the elements. In [52], the authors partition Alu repeats into 213 subfamilies by recursively splitting subfamilies whose members fail a statistical uniformity test. They look for pairs of non-consensus nucleotide values at distinct positions. This allows them to find nested subfamilies (which is impossible using the method of [45]). Finally, they build an evolutionary tree of the subfamilies by computing a minimum spanning tree (MST) with respect to the Hamming distance between subfamily consensus sequences.

In [48], the authors present a randomized method for computing the most likely phylogeny of large sets ($\sim$1,000,000) of mobile elements. The authors assume that only a small number of the elements actively replicate and that all the resultant copies are highly similar on the sequence level and, therefore, elude distance-based clustering methods. Their method

partitions the elements using a randomized clustering algorithm (not based on EM due to its slow convergence). It is an extension of the method presented in [52]; the recursive splitting into subfamilies is done by randomly testing pairs of positions for correlation (in lieu of exhaustive testing), requiring only time that is linear in the repeat sequence length.

In Chapter 3, we use duplication distance to analyze a set of regions of the human genome that contain segmental duplications. First, we find the most parsimonious duplication scenario consistent with the so-called two-step model of segmental duplication using duplication distance as our measure of parsimony. We then refine our notion of a duplication history and compute duplication history DAGs for the regions containing segmental duplications using both a parsimony and a likelihood criterion.

DUPLICATION DISTANCE: A COMBINATORIAL MODEL OF SEGMENTAL DUPLICATIONS

In this chapter, we introduce a novel measure of similarity between genomic regions containing repeated elements, *duplication distance*. We begin by reviewing some definitions and notation that were introduced in [36] and [37]. Let $\emptyset$ denote the empty string. For a string $\mathbf{x} = x_1 \dots x_n$, let $\mathbf{x}_{i,j}$ denote the substring $x_i x_{i+1} \dots x_j$. We define a *subsequence* S of $\mathbf{x}$ to be a string $x_{i_1} x_{i_2} \dots x_{i_k}$ with $i_1 < i_2 < \cdots < i_k$. We represent S by listing the indices at which the characters of S occur in $\mathbf{x}$. For example, if $\mathbf{x} = abcdef$, then the subsequence $S = (1, 3, 5)$ is the string ace. Note that every substring is a subsequence, but a subsequence need not be a substring since the characters comprising a subsequence need not be contiguous. For a pair of subsequences S_1, S_2, denote by $S_1 \cap S_2$ the maximal subsequence common to both S_1 and S_2.

Definition 1. *Subsequences $S = (s_1, s_2)$ and $T = (t_1, t_2)$ of a string $\mathbf{x}$ are **alternating** in $\mathbf{x}$ if either $s_1 < t_1 < s_2 < t_2$ or $t_1 < s_1 < t_2 < s_2$.*

Definition 2. *Subsequences $S = (s_1, \dots, s_k)$ and $T = (t_1, \dots, t_l)$ of a string $\mathbf{x}$ are **overlapping** in $\mathbf{x}$ if there exist indices i, i' and j, j' such that $1 \leq i < i' \leq k$, $1 \leq j < j' \leq l$, and $(s_i, s_{i'})$ and $(t_j, t_{j'})$ are alternating in $\mathbf{x}$. See Fig. 2.1.*

Definition 3. *Given subsequences $S = (s_1, \dots, s_k)$ and $T = (t_1, \dots, t_l)$ of a string $\mathbf{x}$, S is **inside** of T if there exists an index i such that $1 \leq i < l$ and $t_i < s_1 < s_k < t_{i+1}$. That is, the entire subsequence S occurs in between successive characters of T. See Fig. 2.2.*

Definition 4. *A **duplicate operation** from $\mathbf{x}$, $\delta_{\mathbf{x}}(s, t, p)$, copies a substring $x_s \dots x_t$ of the source string $\mathbf{x}$ and pastes it into a target string at position p. Specifically, if $\mathbf{x} = x_1 \dots x_m$ and $Z = z_1 \dots z_n$, then*

$Z \circ \delta_{\mathbf{x}}(s, t, p) = z_1 \dots z_{p-1} x_s \dots x_t z_p \dots z_n$. See Fig. 2.3.

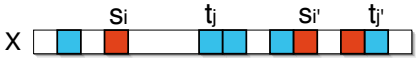

Figure 2.1: The red subsequence is overlapping with the blue subsequence in **x**. The indices $(s_i, s_{i'})$ and $(t_j, t_{j'})$ are alternating in **x**.

Figure 2.2: The red subsequence is inside the blue subsequence T. All the characters of the red subsequence occur between the indices t_i and t_{i+1} of T.

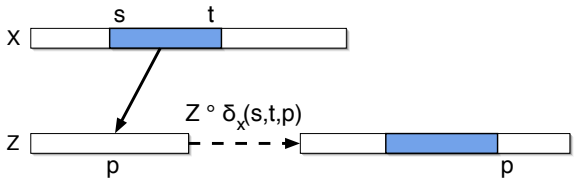

Figure 2.3: A duplicate operation, $\delta_{\mathbf{x}}(s, t, p)$. A substring $x_s x_{s+1} \dots x_t$ of the source string **x** is copied and inserted into the target string Z at index p.

Definition 5. *The **duplication distance** from a source string **x** to a target string **y** is the minimum number of duplicate operations from **x** that generates **y** from an initially empty target string* [1]*. That is,* $\mathbf{y} = \emptyset \circ \delta_{\mathbf{x}}(s_1, t_1, p_1) \circ \delta_{\mathbf{x}}(s_2, t_2, p_2) \circ \cdots \circ \delta_{\mathbf{x}}(s_l, t_l, p_l)$.

2.1 The Basic Recurrence

In this section we review the basic recurrence for computing duplication distance that was introduced in [37]. The recurrence examines the characters of the target string, **y**, and considers the sets of characters of **y** that could have been *generated*, or copied from the source string in a single duplicate operation. Such a set of characters of **y** necessarily correspond to a substring of the source **x** (see Def. 4). Moreover, these characters must be a subsequence of **y**. This is because, in a sequence of duplicate operations, once a string is copied and inserted into the target string, subsequent duplicate operations do not affect the order of the characters in the previously inserted string. Because every character of **y** is generated by exactly one duplicate operation, a sequence of duplicate operations that generates **y** partitions the characters of **y** into disjoint subsequences, each of which is generated in a single duplicate operation. A more interesting observation is that these subsequences are mutually non-overlapping. We formalize this property as follows.

Lemma 1 (Non-overlapping Property)**.** *Consider a source string **x** and a sequence of duplicate operations of the form* $\delta_{\mathbf{x}}(s_i, t_i, p_i)$ *that generates the final target string **y** from an*

[1] We assume that every character in **y** appears at least once in **x**.

*initially empty target string. The substrings x_{s_i,t_i} of **x** that are duplicated during the construction of **y** appear as mutually non-overlapping subsequences of **y**.*

Proof: Consider a sequence of duplicate operations $\delta_{\mathbf{x}}(s_1, t_1, p_1), \ldots, \delta_{\mathbf{x}}(s_k, t_k, p_k)$ that generates **y** from an initially empty target string. For $1 \leq i \leq k$, Let Z^i be the intermediate target string that results from $\delta_{\mathbf{x}}(s_1, t_1, p_1) \circ \cdots \circ \delta_{\mathbf{x}}(s_i, t_i, p_i)$. Note that $Z^k = \mathbf{y}$. For $j \leq i$, let S^i_j be the subsequence of Z^i that corresponds to the characters duplicated by the j^{th} operation. We shall show by induction on the length i of the sequence that that $S^i_1, S^i_2, \ldots, S^i_i$ are pairwise non-overlapping subsequences of Z^i. For the base case, when there is a single duplicate operation, there is no non-overlap property to show. Assume now that $S^{i-1}_1, \ldots S^{i-1}_{i-1}$ are mutually non-overlapping subsequences in Z^{i-1}. For the induction step note that, by the definition of a duplicate operation, S_i is inserted as a contiguous substring into Z^{i-1} at location p_i to form Z^i. Therefore, for any $j, j' < i$, if S^{i-1}_j and $S^{i-1}_{j'}$ are non overlapping in Z^{i-1} then S^i_j and $S^i_{j'}$ are non overlapping in Z^i. It remains to show that for any $j < i$ S^i_j and S^i_i are non-overlapping in Z^i. There are two cases: (1) the elements of S^i_j are either all smaller or all greater than the elements of S^i_i or (2) S^i_i is inside of S^i_j in Z^i (Definition 3). In either case, S_j and S_i are not overlapping in Z^i as required. ■

The non-overlapping property leads to an efficient recurrence that computes duplication distance. When considering subsequences of the final target string **y** that might have been generated in a single duplicate operation, we rely on the non-overlapping property to identify substrings of **y** that can be treated as independent subproblems. If we assume that some subsequence S of **y** is produced in a single duplicate operation, then we know that all other subsequences of **y** that correspond to duplicate operations cannot overlap the characters in S. Therefore, the substrings of **y** in between successive characters of S define subproblems that are computed independently.

In order to find the optimal (i.e. minimum) sequence of duplicate operations that generate **y**, we must consider all subsequences of **y** that could have been generated by a single duplicate operation. The recurrence is based on the observation that y_1 must be the first (i.e. leftmost) character to be copied from **x** in some duplicate operation. There are then two cases to consider: either (1) y_1 was the last (or rightmost) character in the substring that was duplicated from **x** to generate y_1, or (2) y_1 was not the last character in the substring that was duplicated from **x** to generate y_1.

The recurrence defines two quantities: $d(\mathbf{x}, \mathbf{y})$ and $d_i(\mathbf{x}, \mathbf{y})$. We shall show, by induction,

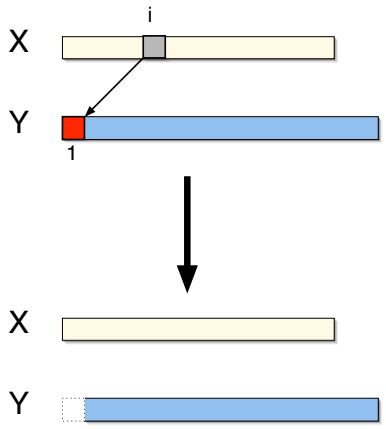

Figure 2.4: y_1 is generated from x_i in a duplicate operation where y_1 is the last (rightmost) character in the copied substring (Case 1). The total duplication distance is one plus the duplication distance for the suffix $\mathbf{y}_{2,|\mathbf{y}|}$.

that for a pair of strings, **x** and **y**, the value $d(\mathbf{x}, \mathbf{y})$ is equal to the duplication distance from **x** to **y** and that $d_i(\mathbf{x}, \mathbf{y})$ is equal to the duplication distance from **x** to **y** under the restriction that the character y_1 is copied from index i in **x**, i.e. x_i *generates* y_1. $d(\mathbf{x}, \mathbf{y})$ is found by considering the minimum among all characters x_i of **x** that can generate y_1, see Eq. 2.1.

As described above, we must consider two possibilities in order to compute $d_i(\mathbf{x}, \mathbf{y})$. Either:

Case 1 : y_1 was the last (or rightmost) character in the substring of **x** that was copied to produce y_1, (see Fig. 2.4), or

Case 2 : x_{i+1} is also copied in the same duplicate operation as x_i, possibly along with other characters as well (see Fig. 2.5).

For case one, the minimum number of duplicate operations is one – for the duplicate that generates y_1 – plus the minimum number of duplicate operations to generate the suffix of **y**, giving a total of $1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$ (Fig. 4). For case two, Lemma 1 implies that the minimum number of duplicate operations is the sum of the optimal numbers of operations for two independent subproblems. Specifically, for each $j > 1$ such that $x_{i+1} = y_j$ we compute: (i) the minimum number of duplicate operations needed to build the substring $\mathbf{y}_{2,j-1}$, namely $d(\mathbf{x}, \mathbf{y}_{2,j-1})$, and (ii) the minimum number of duplicate operations needed to build the string $y_1\mathbf{y}_{j,|\mathbf{y}|}$, given that y_1 is generated by x_i and y_j is generated by x_{i+1}. To compute the latter, recall that since x_i and x_{i+1} are copied in the same duplicate operation, the number of duplicates necessary to generate $y_1\mathbf{y}_{j,|\mathbf{y}|}$ using x_i and x_{i+1} is equal to the number of duplicates necessary to generate $\mathbf{y}_{j,|\mathbf{y}|}$ using x_{i+1}, namely $d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$, (see

Fig. 2.5 and Eq. 2.2).

The recurrence is, therefore:

$$
\begin{aligned}
d(\mathbf{x}, \emptyset) &= 0 \\
d(\mathbf{x}, \mathbf{y}) &= \min_{\{i:x_i=y_1\}} d_i(\mathbf{x}, \mathbf{y}) \qquad (2.1) \\
d_i(\mathbf{x}, \emptyset) &= 0 \\
d_i(\mathbf{x}, \mathbf{y}) &= \min \begin{cases} 1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}) \\ \min_{\{j:y_j=x_{i+1}, j>1\}} \{d(\mathbf{x}, \mathbf{y}_{2,j-1}) + d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})\} \end{cases} \qquad (2.2)
\end{aligned}
$$

Theorem 1. *$d(\boldsymbol{x}, \boldsymbol{y})$ is the minimum number of duplicate operations that generate **y** from **x**. For $\{i : x_i = y_1\}$, $d_i(\boldsymbol{x}, \boldsymbol{y})$ is the minimum number of duplicate operations that generate **y** from **x** such that y_1 is generated by x_i.*

Proof: Let $OPT(\mathbf{x}, \mathbf{y})$ denote minimum length of a sequence of duplicate operations that generate **y** from **x**. Let $OPT_i(\mathbf{x}, \mathbf{y})$ denote the minimum length of a sequence of operations that generate **y** from **x** such that y_1 is generated by x_i. We prove by induction on $\mid \mathbf{y} \mid$ that $d(\mathbf{x}, \mathbf{y}) = OPT(\mathbf{x}, \mathbf{y})$ and $d_i(\mathbf{x}, \mathbf{y}) = OPT_i(\mathbf{x}, \mathbf{y})$.

For $\mid \mathbf{y} \mid = 1$, since we assume there is at least one i for which $x_i = y_1$, $OPT(\mathbf{x}, \mathbf{y}) = OPT_i(\mathbf{x}, \mathbf{y}) = 1$. By definition, the recurrence also evaluates to 1. For the inductive step,

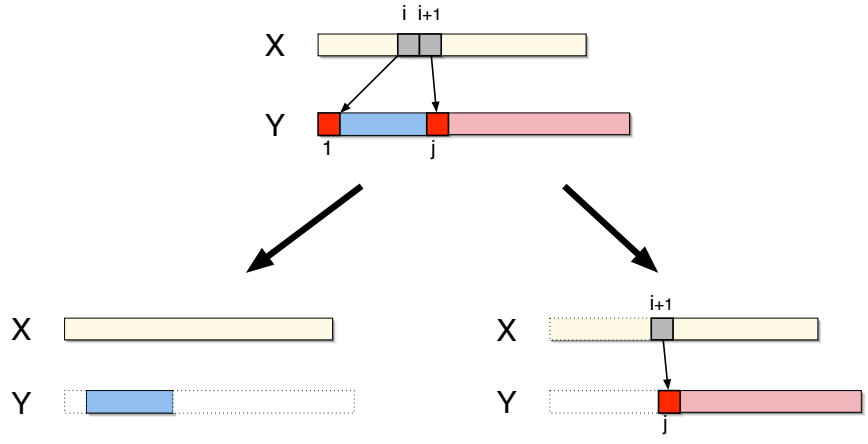

Figure 2.5: y_1 is generated from x_i in a duplicate operation where y_1 is not the last (rightmost) character in a copied substring (Case 2). In this case, x_{i+1} is also copied in the same duplicate operation (top). Thus, the duplication distance is the sum of $d(\mathbf{x}, \mathbf{y}_{2,j-1})$, the duplication distance for $\mathbf{y}_{2,j-1}$ (bottom left), and $d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$, the minimum number of duplicate operations to generate $\mathbf{y}_{j,|\mathbf{y}|}$ given that x_{i+1} generates y_j (bottom right).

assume that $OPT(\mathbf{x}, \mathbf{y}') = d(\mathbf{x}, \mathbf{y}')$ and $OPT_i(\mathbf{x}, \mathbf{y}') = d_i(\mathbf{x}, \mathbf{y}')$ for any string $\mathbf{y}'$ shorter than $\mathbf{y}$. We first show that $OPT_i(\mathbf{x}, \mathbf{y}) \leq d_i(\mathbf{x}, \mathbf{y})$. Since $OPT(\mathbf{x}, \mathbf{y}) = \min_i OPT_i(\mathbf{x}, \mathbf{y})$, this also implies $OPT(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{y})$. We describe different sequences of duplicate operations that generate $\mathbf{y}$ from $\mathbf{x}$, using x_i to generate y_1:

- Consider a minimum-length sequence of duplicates that generates $\mathbf{y}_{2,|\mathbf{y}|}$. By the inductive hypothesis its length is $d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$. By duplicating y_1 separately using x_i we obtain a sequence of duplicates that generates $\mathbf{y}$ whose length is $1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$.
- For every $\{j : y_j = x_{i+1}, j > 1\}$ consider a minimum-length sequence of duplicates that generates $\mathbf{y}_{j,|\mathbf{y}|}$ using x_{i+1} to produce y_j, and a minimum-length sequence of duplicates that generates $\mathbf{y}_{2,j-1}$. By the inductive hypothesis their lengths are $d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$ and $d(\mathbf{x}, \mathbf{y}_{2,j-1})$ respectively. By extending the start index s of the duplicate operation that starts with x_{i+1} to produce y_j to start with x_i and produce y_1 as well, we produce $\mathbf{y}$ with the same number of duplicate operations.

Since $OPT_i(\mathbf{x}, \mathbf{y})$ is at most the length of any of these options, it is also at most their minimum. Hence,

$$\begin{aligned} OPT_i(\mathbf{x}, \mathbf{y}) &\leq \min \begin{cases} 1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}) \\ \min_{\{j: y_j = x_{i+1}, j>1\}} \{d(\mathbf{x}, \mathbf{y}_{2,j-1}) + d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})\} \end{cases} \\ &= d_i(\mathbf{x}, \mathbf{y}). \end{aligned}$$

To show the other direction (i.e. that $d(x, y) \leq OPT(x, y)$ and $d_i(x, y) \leq OPT_i(x, y)$), consider a minimum-length sequence of duplicate operations that generate $\mathbf{y}$ from $\mathbf{x}$, using x_i to generate y_1. There are a few cases:

- If y_1 is generated by a duplicate operation that only duplicates x_i, then $OPT_i(\mathbf{x}, \mathbf{y}) = 1 + OPT(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$. By the inductive hypothesis this equals $1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$ which is at least $d_i(\mathbf{x}, \mathbf{y})$.
- Otherwise, y_1 is generated by a duplicate operation that copies x_i and also duplicates x_{i+1} to generate some character y_j. In this case the sequence Δ of duplicates that generates $\mathbf{y}_{2,j-1}$ must appear after the duplicate operation that generates y_1 and y_j because $\mathbf{y}_{2,j-1}$ is inside (Definition 3) of (y_1, y_j). Without loss of generality, suppose Δ is ordered after all the other duplicates so that first $y_1 y_j \dots y_{|\mathbf{y}|}$ is generated, and then Δ generates $y_2 \dots y_{j-1}$ between y_1 and y_j. Hence, $OPT_i(\mathbf{x}, \mathbf{y}) = OPT_i(\mathbf{x}, y_1 \mathbf{y}_{j,|\mathbf{y}|}) + OPT(\mathbf{x}, \mathbf{y}_{2,j-1})$. Since in the optimal sequence x_i generates y_1 in

the same duplicate operation that generates y_j from x_{i+1}, we have $OPT_i(\mathbf{x}, y_1\mathbf{y}_{j,|\mathbf{y}|}) = OPT_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$. By the inductive hypothesis, $OPT(\mathbf{x}, \mathbf{y}_{2,j-1}) + OPT_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) = d(\mathbf{x}, \mathbf{y}_{2,j-1}) + d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$ which is at least $d_i(\mathbf{x}, \mathbf{y})$.

■

This recurrence naturally translates into a dynamic programing algorithm that computes the values of $d(\mathbf{x}, \cdot)$ and $d_i(\mathbf{x}, \cdot)$ for various target strings. To analyze the running time of this algorithm, note that both $\mathbf{y}_{2,j}$ and $\mathbf{y}_{j,|\mathbf{y}|}$ are substrings of $\mathbf{y}$. Since the set of substrings of $\mathbf{y}$ is closed under taking substrings, we only encounter substrings of $\mathbf{y}$. Also note that since i is chosen from the set $\{i : x_i = y_1\}$, there are $O(\mu(\mathbf{x}))$ choices for i, where $\mu(\mathbf{x})$ is the maximal multiplicity of a character in $\mathbf{x}$. Thus, there are $O(\mu(\mathbf{x}) \mid \mathbf{y} \mid^2)$ different values to compute. Each value is computed by considering the minimization over at most $\mu(\mathbf{y})$ previously computed values, so the total running time is bounded by $O(\mid \mathbf{y} \mid^2 \mu(\mathbf{x})\mu(\mathbf{y}))$, which is $O(\mid \mathbf{y} \mid^3 \mid \mathbf{x} \mid)$ in the worst case. We note that for applications where the size of the alphabet on which the strings are built is large with respect to the length of the strings, such that $\mu(\mathbf{x}) \in O(1)$ and $\mu(\mathbf{y}) \in O(1)$, the running time of the algorithm is $O(\mid \mathbf{y} \mid^2)$ in the worst case. As with most dynamic programming approaches, this algorithm (and all others presented in subsequent sections) can be extended through trace-back to reconstruct the optimal sequence of operations needed to build $\mathbf{y}$. We omit the details.

2.2 Extending to Affine Duplication Cost

It is easy to extend the recurrence relations in Eqs. (2.1), (2.2) to handle costs for duplicate operations. In the above discussion, the cost of each duplicate operation is 1, so the sum of costs of the operations in a sequence that generates a string $\mathbf{y}$ is just the length of that sequence. We next consider a more general cost model for duplication in which the cost of a duplicate operation $\delta_x(s, t, p)$ is $\Delta_1 + (t - s + 1)\Delta_2$ (i.e., the cost is affine in the number of duplicated characters). Here Δ_1, Δ_2 are some non-negative constants. This extension is obtained by assigning a cost of Δ_2 to each duplicated character, except for the last character in the duplicated string, which is assigned a cost of $\Delta_1 + \Delta_2$. We do that by adding a cost term to each of the cases in Eq. 2.2. If x_i is the last character in the duplicated string (case 1), we add $\Delta_1 + \Delta_2$ to the cost. Otherwise x_i is not the last duplicated character (case 2),

so we add just Δ_2 to the cost. Eq. (2.2) thus becomes

$$d_i(\mathbf{x}, \mathbf{y}) \;=\; \min \begin{cases} \Delta_1 + \Delta_2 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}) \\ \min_{\{j: y_j = x_{i+1}, j>1\}} \{ d(\mathbf{x}, \mathbf{y}_{2,j-1}) + d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) + \Delta_2 \} \end{cases} \tag{2.3}$$

The running time analysis for this recurrence is the same as for the one with unit duplication cost.

2.3 Extending the Model: Duplication-Deletion Distance

Here we provide several extensions to the duplication distance model; we generalize the model to allow also for certain types of substring deletions in the target string.

Consider the intermediate string Z generated after some number of duplicate operations. A deletion operation removes a contiguous substring $z_i, \ldots, z_j$ of Z, and subsequent duplicate and deletion operations are applied to the resulting string.

Definition 6. *A **delete operation**, $\tau(s,t)$, deletes a substring $z_s \ldots z_t$ of the target string Z, thus making Z shorter. Specifically, if $Z = z_1 \ldots z_s \ldots z_t \ldots z_m$, then $Z \circ \tau(s,t) = z_1 \ldots z_{s-1} z_{t+1} \ldots z_m$. See Figure 6.*

The cost associated with $\tau(s,t)$ depends on the number $t - s + 1$ of characters deleted and is denoted $\Phi(t - s + 1)$.

Figure 2.6: A delete operation, $\tau(s,t)$. The substring $Z_{s,t}$ is deleted.

Definition 7. *The **duplication-deletion distance** from a source string **x** to a target string **y** is the cost of a minimum sequence of duplicate operations from **x** and deletion operations, in any order, that generates **y**.*

We now show that although we allow arbitrary deletions from the intermediate string, it suffices to consider deletions from the duplicated strings before they are pasted into the intermediate string, provided that the cost function for deletion, $\Phi(\cdot)$ is non-decreasing and obeys the triangle inequality.

Definition 8. *A **duplicate-delete** operation from **x**, $\eta_{\mathbf{x}}(i_1, j_1, i_2, j_2, \ldots, i_k, j_k, p)$, for $i_1 \leq j_1 < i_2 \leq j_2 < \cdots < i_k \leq j_k$ copies the subsequence $x_{i_1} \ldots x_{j_1} x_{i_2} \ldots x_{j_2} \ldots\ldots x_{i_k} \ldots x_{j_k}$ of the source string **x** and pastes it into a target string at position p. Specifically, if $\mathbf{x} = x_1 \ldots x_m$ and $Z = z_1 \ldots z_n$, then $Z \circ \eta_{\mathbf{x}}(i_1, j_1, \ldots, i_k, j_k, p) =$*

$z_1 \dots z_{p-1} x_{i_1} \dots x_{j_1} x_{i_2} \dots x_{j_2} \dots x_{i_k} \dots x_{j_k} z_p \dots z_n$.

The cost associated with such a duplicate-delete is $\Delta_1 + (j_k - i_1 + 1)\Delta_2 + \sum_{\ell=1}^{k-1} \Phi(i_{\ell+1} - j_\ell - 1)$. The first two terms in the cost reflect the affine cost of duplicating an entire substring of length $j_k - i_1 + 1$, and the second term reflects the cost of deletions made to that substrings.

Lemma 2. *If the affine cost for duplications is non-decreasing and $\Phi(\cdot)$ is non-decreasing and obeys the triangle inequality then the cost of a minimum sequence of duplicate and delete operations that generates a target string **y** from a source string **x** is equal to the cost of a minimum sequence of duplicate-delete operations that generates **y** from **x**.*

Proof: Since duplicate operations are a special case of duplicate-delete operations, the cost of a minimal sequence of duplicate-delete operations and delete operations that generates **y** cannot be more than that of a sequence of just duplicate operations and delete operations. We show the (stronger) claim that an arbitrary sequence of duplicate-delete and delete operations that produces a string **y** with cost c can be transformed into a sequence of just duplicate-delete operations that generates **y** with cost at most c by induction on the number of delete operations. The base case, where the number of deletions is zero, is trivial. Consider the first delete operation, τ. Let k denote the number of duplicate-delete operations that precede τ, and let Z be the intermediate string produced by these k operations. For $i = 1, \dots, k$, let S_i be the subsequence of **x** that was used in the ith duplicate-delete operation. By lemma 1, $S_1, \dots, S_k$ form a partition of Z into disjoint, non-overlapping subsequences of Z. Let d denote the substring of Z to be deleted. Since d is a contiguous substring, $S_i \cap d$ is a (possibly empty) substring of S_i for each i. There are several cases:

1. $S_i \cap d = \emptyset$. In this case we do not change any operation.

2. $S_i \cap d = S_i$. In this case all characters produced by the ith duplicate-delete operation are deleted, so we may omit the ith operation altogether and decrease the number of characters deleted by τ. Since $\Phi(\cdot)$ is non-decreasing, this does not increase the cost of generating Z (and hence **y**).

3. $S_i \cap d$ is a prefix (or suffix) of S_i. Assume it is a prefix. The case of suffix is similar. Instead of deleting the characters $S_i \cap d$ we can avoid generating them in the first place. Let r be the smallest index in $S_i \setminus d$ (that is, the first character in S_i that is not deleted by τ). We change the ith duplicate-delete operation to start at r and decrease the number of characters deleted by τ. Since the affine cost for duplications is non-decreasing and $\Phi(\cdot)$ is non-decreasing, the cost of generating Z does not increase.

4. $S_i \cap d$ is a non-empty substring of S_i that is neither a prefix nor a suffix of S_i. We claim that this case applies to at most one value of i. This implies that after taking care of all the other cases τ only deletes characters in S_i. We then change the ith duplicate-delete operation to also delete the characters deleted by τ, and omit τ. Since $\Phi(\cdot)$ obeys the triangle inequality, this will not increase the total cost of deletion. By the inductive hypothesis, the rest of $\mathbf{y}$ can be generated by just duplicate-delete operations with at most the same cost. It remains to prove the claim. Recall that the set $\{S_i\}$ is comprised of mutually non-overlapping subsequences of Z. Suppose that there exist indices $i \neq j$ such that $S_i \cap d$ is a non-prefix/suffix substring of S_i and $S_j \cap d$ is a non-prefix/suffix substring of S_j. There must exist indices of both S_i and S_j in Z that precede d, are contained in d, and succeed d. Let $i_p < i_c < i_s$ be three such indices of S_i and let $j_p < j_c < j_s$ be similar for S_j. It must be the case also that $j_p < i_c < j_s$ and $i_p < j_c < i_s$. Without loss of generality, suppose $i_p < j_p$. It follows that (i_p, i_c) and (j_p, j_s) are alternating in Z. So, S_i and S_j are overlapping which contradicts Lemma 1.

■

To extend the basic recurrence to duplication-deletion distance, we must observe that because we allow deletions in the string that is duplicated from $\mathbf{x}$, if we assume character x_i is copied to produce y_1, it may not be the case that the character x_{i+1} also appears in $\mathbf{y}$; the character x_{i+1} may have been deleted. Therefore, we minimize over all possible locations $k > i$ for the next character in the duplicated string that is not deleted. The extension of the recurrence from the previous section to duplication-deletion distance is:

$$\hat{d}(\mathbf{x}, \emptyset) = 0 \quad , \quad \hat{d}(\mathbf{x}, \mathbf{y}) = \min_{\{i: x_i = y_1\}} \hat{d}_i(\mathbf{x}, \mathbf{y}), \tag{2.4}$$

$$\hat{d}_i(\mathbf{x}, \emptyset) = 0 \quad ,$$

$$\hat{d}_i(\mathbf{x}, \mathbf{y}) = \min \left\{ \begin{array}{l} \Delta_1 + \Delta_2 + \hat{d}(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{k>i} \min_{\{j: y_j = x_k, j>1\}} \left\{ \begin{array}{l} \hat{d}(\mathbf{x}, \mathbf{y}_{2,j-1}) + \hat{d}_k(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) \\ (k-i)\Delta_2 + \Phi(k-i-1) \end{array} \right\}. \end{array} \right. \tag{2.5}$$

Theorem 2. *$\hat{d}(\boldsymbol{x}, \boldsymbol{y})$ is the duplication-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$. For $\{i : x_i = y_1\}$, $\hat{d}_i(\boldsymbol{x}, \boldsymbol{y})$ is the duplication-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$ under the additional restriction that y_1 is generated by x_i.*

The proof of Theorem 2 is an extension to that of Theorem 1. However, the running time increases; while the number of entries in the dynamic programming table does not change,

the time to compute each entry is multiplied by the possible values of k in the recurrence, which is $O(\mid \mathbf{x} \mid)$. Therefore, the running time is $O(\mid \mathbf{y} \mid^2 \mid \mathbf{x} \mid \mu(\mathbf{x})\mu(\mathbf{y}))$, which is $O(\mid \mathbf{y} \mid^3 \mid \mathbf{x} \mid^2)$ in the worst case.

We now show, in the following lemma, that if both the duplicate and delete cost functions are the identity function (i.e. one per operation), then the duplication-deletion distance is equal to duplication distance without deletions.

Lemma 3. *Given a source string* $\boldsymbol{x}$*, a target string* $\boldsymbol{y}$*, If the cost of duplication is 1 per duplicate operation, and the cost of deletion is 1 per delete operation, then* $\hat{d}(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{x}, \boldsymbol{y})$.

Proof: First we note that if a target string $\mathbf{y}$ can be built from $\mathbf{x}$ in $d(\mathbf{x}, \mathbf{y})$ duplicate operations, then the same sequence of duplicate operations is a valid sequence of duplicate and delete operations as well, so $d(\mathbf{x}, \mathbf{y})$ is at least $\hat{d}(\mathbf{x}, \mathbf{y})$.

We claim that every sequence of duplicate and delete operations can be transformed into a sequence of duplicate operations of the same length. The proof of this claim is similar to that of Lemma 2. In that proof we showed how to transform a sequence of duplicate and delete operations into a sequence of duplicate-delete operations of at most the same cost. We follow the same steps, but transform the sequence into an a sequence that consists of just duplicate operations without increasing the number of operations. Recall the four cases in the proof of Lemma 2. In the the first three cases we eliminate the delete operation without increasing the number of duplicate operations. Therefore we only need to consider the last case ($S_i \cap d$ is a non-empty substring of S_i that is neither a prefix nor a suffix of S_i). Recall that this case applies to at most one value of i. Deleting $S_i \cap d$ from S_i leaves a prefix and a suffix of S_i. We can therefore replace the i^{th} duplicate operation and the delete operation with two duplicate operations, one generating the appropriate prefix of S_i and the other generating the appropriate suffix of S_i. This eliminates the delete operation without changing the number of operations in the sequence. Therefore, for any string $\mathbf{y}$ that results from a sequence of duplicate and delete operations, we can construct the same string using only duplicate operations (without deletes) using at most the same number of operations. So, $d(\mathbf{x}, \mathbf{y})$ is no greater than $\hat{d}(\mathbf{x}, \mathbf{y})$. ■

2.4 Extending the Model: Duplication-Inversion Distance

Here we generalize the duplication distance model to allow also for substring inversions. We now explicitly define characters and strings as having two orientations: forward (+) and

inverse (-).

Definition 9. *A **signed string** of length m over an alphabet Σ is an element of $(\{+,-\} \times \Sigma)^m$.*

For example, $(+b\ -c\ -a\ +d)$ is a signed string of length 4. An inversion of a signed string reverses the order of the characters as well as their signs. Formally,

Definition 10. *The **inverse** of a signed string $\boldsymbol{x} = x_1 \dots x_m$ is a signed string $\bar{\boldsymbol{x}} = -x_m\ \dots\ -x_1$.*

For example, the inverse of $(+b\ -c\ -a\ +d)$ is $(-d\ +a\ +c\ -b)$.

In a duplicate-invert operation a substring is copied from **x** and *inverted* before being inserted into the target string **y**. We allow the cost of inversion to be an affine function in the length ℓ of the duplicated inverted string, which we denote $\Theta_1 + \ell\Theta_2$, where $\Theta_1, \Theta_2 \geq 0$. We still allow for normal duplicate operations.

Definition 11. *A **duplicate-invert operation** from $\boldsymbol{x}$, $\bar{\delta}_{\boldsymbol{x}}(s,t,p)$, copies an inverted substring $-x_t,\ -x_{t-1}\ \dots,\ -x_s$ of the source string $\boldsymbol{x}$ and pastes it into a target string at position p. Specifically, if $\boldsymbol{x} = x_1 \dots x_m$ and $\mathbf{Z}_{=} z_1 \dots z_n$, then $\mathbf{Z}_{\circ}\bar{\delta}_{\boldsymbol{x}}(s,t,p) =$*
$z_1 \dots z_{p-1}\bar{x}_t\bar{x}_{t-1} \dots \bar{x}_s z_p \dots z_n$.

The cost associated with each duplicate-invert operation is $\Theta_1 + (t-s+1)\Theta_2$.

Definition 12. *The **duplication-inversion distance** from a source string $\boldsymbol{x}$ to a target string $\boldsymbol{y}$ is the cost of a minimum sequence of duplicate and duplicate-invert operations from $\boldsymbol{x}$, in any order, that generates $\boldsymbol{y}$.*

The recurrence for duplication distance (Eqs. 2.1, 2.3) can be extended to compute the duplication-inversion distance. This is done by introducing a term for inverted duplications whose form is very similar to that of the term for regular duplication (Eq. 2.3). Specifically, when considering the possible characters to generate y_1, we consider characters in **x** that match either y_1 or its inverse, $-y_1$. In the former case, then, we use $\bar{d}_i^{+}(\mathbf{x}, \mathbf{y})$ to denote the duplication-inversion distance with the additional restriction that y_1 is generated by x_i without an inversion. The recurrence for $\bar{d}_i^{+}$ is the same as for d_i in Eq. 2.3. In the latter case, we consider an inverted duplicate in which y_1 is generated by $-x_i$. This is denoted by $\bar{d}_i^{-}$, which follows a similar recurrence. In this recurrence, since an inversion occurs, x_i is the *last* character of the duplicated string, rather than the first one. Therefore, the next character in **x** to be used in this operation is $-x_{i-1}$ rather than x_{i+1}. The recurrence for $\bar{d}_i^{-}$ also differs in the cost term, where we use the affine cost of the duplicate-invert operation.

The extension of the recurrence to duplication-inversion distance is therefore:

$$
\begin{aligned}
\bar{d}(\mathbf{x},\emptyset) &= 0 \quad , \quad \bar{d}(\mathbf{x},\mathbf{y}) = \min\left\{ \min_{\{i:x_i=y_1\}} \bar{d}_i^{+}(\mathbf{x},\mathbf{y}), \min_{\{i:x_i=-y_1\}} \bar{d}_i^{-}(\mathbf{x},\mathbf{y}) \right\}, \\
\bar{d}_i^{+}(\mathbf{x},\emptyset) &= 0 \quad , \quad \bar{d}_i^{-}(\mathbf{x},\emptyset) = 0, \\
\bar{d}_i^{+}(\mathbf{x},\mathbf{y}) &= \min\left\{ \begin{array}{l} \Delta_1 + \Delta_2 + \bar{d}(\mathbf{x},\mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{\{j:y_j=x_{i+1},j>1\}} \left\{ \bar{d}(\mathbf{x},\mathbf{y}_{2,j-1}) + \bar{d}_{i+1}^{+}(\mathbf{x},\mathbf{y}_{j,|\mathbf{y}|}) + \Delta_2 \right\}, \end{array} \right. \\
\bar{d}_i^{-}(\mathbf{x},\mathbf{y}) &= \min\left\{ \begin{array}{l} \Theta_1 + \Theta_2 + \bar{d}(\mathbf{x},\mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{\{j:y_j=-x_{i-1},j>1\}} \left\{ \bar{d}(\mathbf{x},\mathbf{y}_{2,j-1}) + \bar{d}_{i-1}^{-}(\mathbf{x},\mathbf{y}_{j,|\mathbf{y}|}) + \Theta_2 \right\}. \end{array} \right.
\end{aligned}
\tag{2.6}
$$

Theorem 3. *$\bar{d}(\boldsymbol{x},\boldsymbol{y})$ is the duplication-inversion distance from **x** to **y**. For $\{i : x_i = y_1\}$, $\bar{d}_i^{+}(\boldsymbol{x},\boldsymbol{y})$ is the duplication-inversion distance from **x** to **y** under the additional restriction that y_1 is generated by x_i. For $\{i : x_i = -y_1\}$, $\bar{d}_i^{-}(\boldsymbol{x},\boldsymbol{y})$ is the duplication-inversion distance from **x** to **y** under the additional restriction that y_1 is generated by $-x_i$.*

The correctness proof is very similar to that of Theorem 1, only requiring an additional case for handling the case of a duplicate invert operation which is symmetric to the case of regular duplication. The asymptotic running time of the corresponding dynamic programming algorithm is $O(\mid \mathbf{y} \mid^2 \mu(\mathbf{x})\mu(\mathbf{y}))$. The analysis is identical to the one in section **??**. The fact that we now consider either a duplicate or a duplicate-invert operation does not change the asymptotic running time.

2.5 Extending the Model: Duplication-Inversion-Deletion Distance

Here we extend the distance measure to include delete operations as well as duplicate and duplicate-invert operations. Note that we only handle deletions after inversions of the same substring. The order of operations might be important, at least in terms of costs. The cost of inverting $(+a\ +b\ +c)$ and then deleting $-b$ may be different than the cost of first deleting $+b$ from $(+a\ +b\ +c)$ and then inverting $(+a\ +c)$.

Definition 13. *The **duplication-inversion-deletion distance** from a source string **x** to a target string **y** is the cost of a minimum sequence of duplicate and duplicate-invert operations from **x** and deletion operations, in any order, that generates **y**.*

Definition 14. *A **duplicate-invert-delete** operation from **x**, $\bar{\eta}_{\boldsymbol{x}}(i_1, j_1, i_2, j_2, \ldots, i_k, j_k, p)$, for $i_1 \leq j_1 < i_2 \leq j_2 < \cdots < i_k \leq j_k$ pastes the string*

$-x_{j_k}\ -x_{j_k-1}\ \ldots\ -x_{i_k}\ -x_{j_{k-1}}\ -x_{j_{k-1}-1}\ \ldots -x_{i_{k-1}}\ \ldots\ldots -x_{j_1}\ -x_{j_1-1}\ \ldots\ -x_{i_1}$ *into a target string at position* p. *Specifically, if* $\boldsymbol{x} = x_1 \ldots x_m$ *and* $\mathbf{Z}_{=} z_1 \ldots z_n$, *then* $\mathbf{Z}_{\circ} \bar{\eta}_{\boldsymbol{x}}(i_1, j_1, i_2, j_2, \ldots, i_k, j_k, p) = z_1 \ldots z_{p-1} -x_{j_k} -x_{j_k-1} \ldots -x_{i_k} -x_{j_{k-1}} -x_{j_{k-1}-1} \ldots -x_{i_{k-1}} \ldots\ldots -x_{j_1} -x_{j_i-1} \ldots -x_{i_1}\ z_p \ldots z_n$.

The cost of such an operation is $\Theta_1 + (j_k - i_1 + 1)\Theta_2 + \sum_{\ell=1}^{k-1} \Phi(i_{\ell+1} - j_\ell - 1)$. Similar to the previous section, it suffices to consider just duplicate-invert-delete and duplicate-delete operations, rather than duplicate, duplicate-invert and delete operations.

Lemma 4. *If* $\Phi(\cdot)$ *is non-decreasing and obeys the triangle inequality and if the cost of inversion is an affine non-decreasing function as defined above, then the cost of a minimum sequence of duplicate, duplicate-invert and delete operations that generates a target string* $\boldsymbol{y}$ *from a source string* $\boldsymbol{x}$ *is equal to the cost of a minimum sequence of duplicate-delete and duplicate-invert-delete operations that generates* $\boldsymbol{y}$ *from* $\boldsymbol{x}$.

The proof of the lemma is essentially the same as that of Lemma 2. Note that in that proof we did not require all duplicate operations to be from the same string $\mathbf{x}$. Therefore, the arguments in that proof apply to our case, where we can regard some of the duplicates from $\mathbf{x}$ and some from the inverse of $\mathbf{x}$.

The recurrence for duplication-inversion-deletion distance is obtained by combining the recurrences for duplication-deletion (Eq. 2.5) and for duplication-inversion distance (Eq. 2.6). We use separate terms for duplicate-delete operations ($\hat{\bar{d}}_i^{+}$) and for duplicate-invert-delete operations ($\hat{\bar{d}}_i^{-}$). Those terms differ from the terms in Eq. 2.6 in the same way Eq. 2.5 differs from Eq. 2.2; Because of the possible deletion we do not know that x_{i+1} (x_{i-1}) is the next duplicated character. Instead we minimize over all characters later (earlier) than x_i.

The recurrence for duplication-inversion-deletion distance is therefore:

$$
\begin{aligned}
\hat{\bar{d}}(\mathbf{x}, \emptyset) &= 0 \quad , \quad \hat{\bar{d}}(\mathbf{x}, \mathbf{y}) = \min\left\{ \min_{\{i: x_i = y_1\}} \hat{\bar{d}}_i^{+}(\mathbf{x}, \mathbf{y}), \min_{\{i: x_i = -y_1\}} \hat{\bar{d}}_i^{-}(\mathbf{x}, \mathbf{y}) \right\}, \\
\hat{\bar{d}}_i^{+}(\mathbf{x}, \emptyset) &= 0 \quad , \quad \hat{\bar{d}}_i^{-}(\mathbf{x}, \emptyset) = 0, \\
\hat{\bar{d}}_i^{+}(\mathbf{x}, \mathbf{y}) &= \min\left\{ \begin{array}{l} \Delta_1 + \Delta_2 + \hat{\bar{d}}(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{k>i} \min_{\{j: y_j = x_k, j>1\}} \left\{ \begin{array}{l} \hat{\bar{d}}(\mathbf{x}, \mathbf{y}_{2,j-1}) + \hat{\bar{d}}_k^{+}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) \\ (k-i)\Delta_2 + \Phi(k-i-1) \end{array} \right\} \end{array} \right., \\
\hat{\bar{d}}_i^{-}(\mathbf{x}, \mathbf{y}) &= \min\left\{ \begin{array}{l} \Theta_1 + \Theta_2 + \hat{\bar{d}}(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{k<i} \min_{\{j: y_j = -x_k, j>1\}} \left\{ \begin{array}{l} \hat{\bar{d}}(\mathbf{x}, \mathbf{y}_{2,j-1}) + \hat{\bar{d}}_k^{-}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) \\ +(i-k)\Theta_2 + \Phi(i-k-1) \end{array} \right\} \end{array} \right..
\end{aligned}
$$

Theorem 4. *$\hat{\bar{d}}(\boldsymbol{x},\boldsymbol{y})$ is the duplication-inversion-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$. For $\{i : x_i = y_1\}$, $\hat{\bar{d}}_i^{+}(\boldsymbol{x},\boldsymbol{y})$ is the duplication-inversion-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$ under the additional restriction that y_1 is generated by x_i. For $\{i : x_i = -y_1\}$, $\hat{\bar{d}}_i^{-}(\boldsymbol{x},\boldsymbol{y})$ is the duplication-inversion-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$ under the additional restriction that y_1 is generated by $-x_i$.*

The proof, again, is very similar to the proofs in the previous sections. The running time of the corresponding dynamic programming algorithm is the same (asymptotically) as that of duplication-deletion distance. It is $O(\mid \mathbf{y} \mid^2 \mid \mathbf{x} \mid \mu(\mathbf{y})\mu(\mathbf{x}))$, where the multiplicity $\mu(\mathbf{y})$ (or $\mu(\mathbf{x})$) is the number of times a character appears in the string $\mathbf{y}$ (or $\mathbf{x}$), regardless of its sign.

In comparing the models of the previous section and the current one, we note that restricting the model of rearrangement to allow only duplicate and duplicate-invert operations instead of duplicate-invert-delete operations may be desirable from a biological perspective because each duplicate and duplicate-invert requires only three breakpoints in the genome, whereas a duplicate-invert-delete operation can be significantly more complicated, requiring more breakpoints.

ANALYSIS OF HUMAN SEGMENTAL DUPLICATIONS

The duplication distance model we presented in Chapter 2 is inspired by biological models for the emergence of segmental duplications (or low-copy repeats) in mammalian genomes. Approximately 5% of the human genome consists of segmental duplications > 1 kb in length with $\geq 90\%$ sequence identity between copies[6]. Segmental duplications account for a significant fraction of the differences between humans and other primate genomes, and are enriched for genes that are differentially expressed between the species [15]. Human segmental duplications contain novel fusion genes[57], genes under strong positive selection [21], and new gene families[40]. Moreover, the presence of segmental duplications appears to render regions of the genome more susceptible to recurrent and disease-causing rearrangements[42] as well as additional copy-number variants [6] and inversions [61].

Reconstructing the evolutionary history of these genomic regions is a non-trivial, but important task as segmental duplications harbor recent primate-specific and human-specific innovations [31]. Moreover, since segmental duplications arise as copy-number variants that become fixed in a population, the evolutionary history of segmental duplications reveals information about the mechanisms and temporal dynamics of copy-number variants in the human genome [38].

The availability of genome sequences from multiple mammalian genomes has led to proposals to reconstruct the genome sequence of the mammalian ancestor [13]. Segmental duplications remain an extreme challenge for evolutionary reconstruction, as they are the "most structurally complex and dynamic regions of the human genome" [2].

Human segmental duplications are frequently found within complicated mosaics (**duplication blocks**) of duplicated fragments (**duplicons**) that bear sequence similarity to non-homologous

regions on multiple human chromosomes [6, 7, 6]. A **two-step model** of segmental duplication (reviewed in [6]) has been proposed to explain these mosaic patterns in pericentromeric regions. In the two-step model, duplicons from disparate regions of the genome (possibly different chromosomes) are first copied and aggregated in a *seeding event*. Then in a second phase of pericentromeric transfer, contiguous sequences of duplicons are transferred en bloc by duplication to non-homologous pericentromeric regions. The result is that "the pericentromeric region consists of many juxtaposed duplicons that originate from diverse ancestral regions," [6]. By contrast, duplication blocks in subtelomeric regions are thought to arise from the process of double-stranded breakage and repair, resulting in interchromosomal translocations of contiguous subtelomeric regions. Finally, interstitial regions gain duplication blocks as a result of multiple rounds of serial duplication. These three proposed mechanisms for the creation of duplication blocks in the human genome indicates the complexity of these regions. As a result, the convoluted nature of overlapping, interleaved duplicated material in the genome makes segmental duplications refractory to traditional sequence analysis.

Jiang et al. [32] recently produced a comprehensive annotation of this mosaic organization: they derived an "alphabet" of approximately 11,000 duplicons, and identified 437 duplication blocks, or "strings" containing at least 10 (and typically dozens) of different duplicons. They also examined the ancestral relationships between human segmental duplications, and identified "clades" of segmental duplications that share an abundance of repeated subsequences. However, their approach ignored the order and orientation of these repeated subsequences within the segmental duplications, and thus did not explicitly explain the mosaic organization of segmental duplications. The relationships between these annotated duplication blocks are complex (Fig. 3.1) and straightforward analysis does not immediately reveal the ancestral relationships between blocks.

Numerous authors have considered the problem of analyzing relationships between genome sequences that contain duplicated segments. This work falls into roughly two categories. The first focus is the problem of computing genome rearrangement distances, like reversal distance, in the presence of duplicated genes or synteny blocks (see [54, 43, 24], for example). However, such rearrangement distances do not model the creation of new duplicates and thus are not well-suited to describe the evolutionary history of segmental duplications in the genome. The second focus is to analyze regions with duplications under "local" operations like tandem duplications (see [18, 39], for example). While tandem duplication is undoubtedly important in the generation of duplication blocks, there is strong evidence that

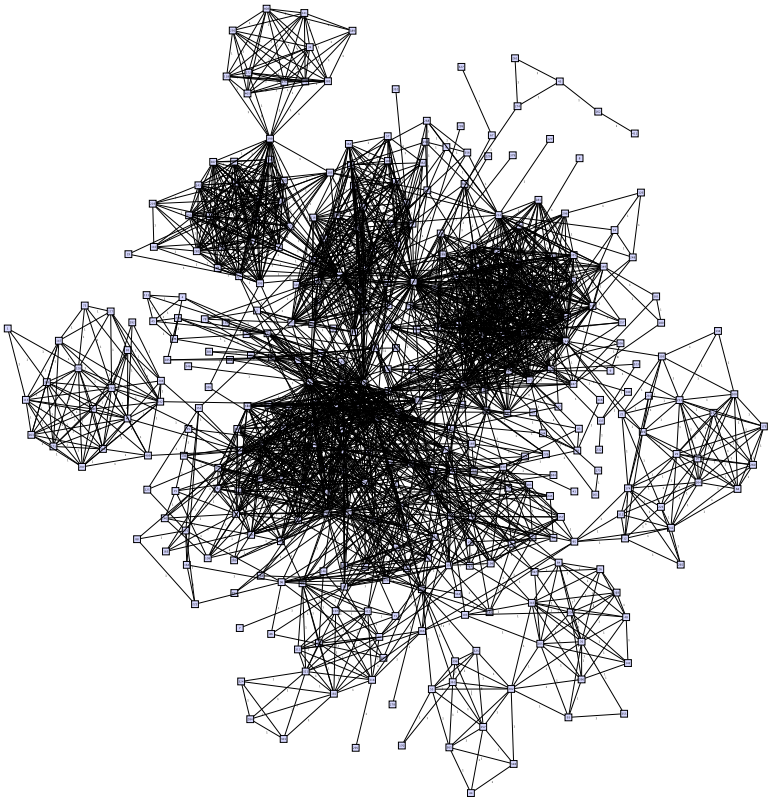

Figure 3.1: A graph of relationships between a subset of 357 duplication blocks in the human genome. Each vertex is a duplication block, with edges joining blocks whose longest common subsequence includes at least 3 duplicons.

an important characteristic of the history of segmental duplications is the frequent duplication and transposition of long segments over large physical distances; as many as 50%-60% of segmental duplications were transposed interchromosomally [6]. Several general models of rearrangement that allowed for both local operations and duplication-transposition-like operations between different strings were studied by [27], but the generality of those models meant that the distances were NP-hard to compute and only approximation algorithms were given.

In this chapter we consider the problem of constructing evolutionary histories that can account for the emergence of the duplication blocks we observe in the present-day human genome. In Section 3.1, we formulate the problem as an integer linear program, inspired by the two-step model of segmental duplication, with an objective function that minimizes the total number of duplicate operations needed to construct all the present-day duplication blocks in a two duplication phases. We represent the resulting evolutionary relationships between duplication blocks as a tree with height two. We use the duplication distance algorithm presented in Chapter 2 to find the optimal tree solution. Then in Section 3.2, we generalize the optimization problem formulation to allow for the construction of an evolutionary history directed acyclic graph (DAG). We define the optimization problem with respect to two criteria: a parsimony criterion that again uses duplication distance as a measure of parsimony and a likelihood criterion that uses a probabilistic model of duplication based on a partition function of the ensemble of all possible duplication scenarios. In both sections, we apply our methods to the analysis of segmental duplications in the human genome using the set of duplication blocks and constituent duplicons annotated in [32].

3.1 The Most-Parsimonious Two-Step Duplication Tree

As described above, duplication blocks, or segments of the present-day genome that contain duplicated material, contain complex mosaic patterns of smaller segments, known as duplicons, that appear in multiplicity across the genome. We model both the ancestral and present-day genomes as signed strings on an alphabet of duplicons. We assume the present-day genome, which has incurred segmental duplications, is a superstring of the ancestral genome, and the duplication blocks are substrings of the present-day genome. See Figure 3.2.

Recall that according to the the two-step model of duplication, duplicons are copied from their ancestral loci and aggregated into larger, contiguous segments or *seed duplication*

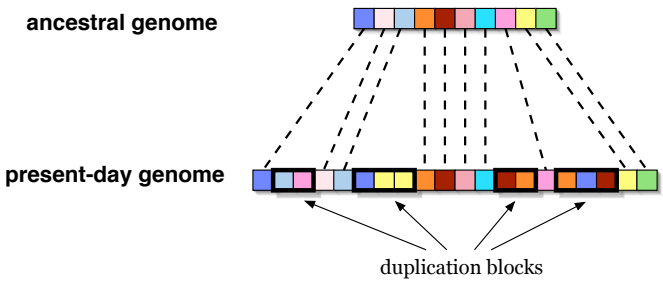

Figure 3.2: The present-day genome is a superstring of the ancestral genome. The duplicated material comprises *duplication blocks* which are maximal contiguous substrings of the present-day genome that were not part of the ancestral genome.

blocks during the first duplication phase. In the second phase, substrings of both the seed blocks and the ancestral genome are copied and then reinserted into the genome at disparate locations, creating *secondary duplication blocks*. We say that a seed duplication block *seeds* a secondary duplication block if substrings of the seed block are used in the construction of the secondary block in the second phase. See Figure 3.3(a). Note that a seed block may seed multiple secondary duplication blocks but there may also exist some seed blocks that do not seed any secondary blocks.

Here we build a duplication scenario that is consistent with a rather literal interpretation of the two-step model of duplication and that minimizes the total number of duplication operations needed to construct a given set of duplication blocks in two phases – first by constructing a set of seed blocks by aggregating duplicons from the ancestral genome and then by constructing a set of secondary blocks by copying substrings of the seed blocks as well as singleton duplicons from their ancestral loci. We formulate this problem as an integer linear program that is equivalent to the facility location problem, a classic problem in operations research. The formulation requires a measure of the minimum number of duplicate operations needed to build a target duplication block from a source duplication block; we use the duplication distance measure described in Chaper 2. We apply our method to duplication blocks derived in [32] and discover a two-step duplication scenario in which 64 seed duplication blocks are first constructed and then duplicated to create secondary duplication blocks.

Note that we make four simplifying assumptions about the two-step model of duplication:

1. The ancestral genome contains exactly one copy of every duplicon.

2. No other type of rearrangement operations – such as inversions or deletions – occur.

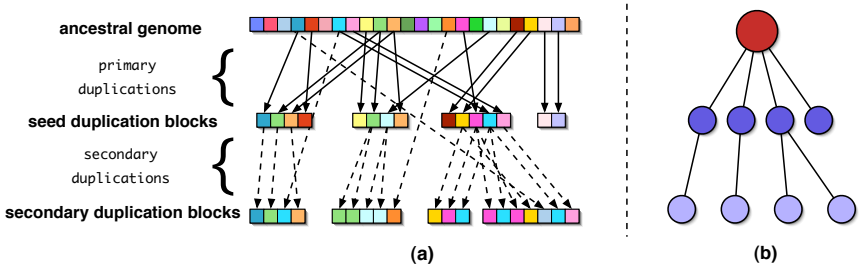

Figure 3.3: **(a)** The two-step model of duplication. Solid arrows indicate duplicons copied during the first phase of duplication. Dashed arrows indicate duplicons copied during the second phase of duplication. **(b)** The corresponding two-step duplication tree.

3. The seed blocks are a subset of the duplication blocks observed in the present-day genome.

4. Each secondary duplication block is seeded by exactly one seed duplication block.

Under these assumptions, we can describe a duplication scenario in which a set of duplication blocks are constructed in two phases by representing it as a *two-step duplication tree* on the set of duplication blocks.

Definition 15. *Given ancestral and present-day genomes, a* ***two-step duplication tree*** *is a tree of height three where the root is the ancestral genome and the descendants are the duplication blocks. Nodes at depth one (i.e. the children of the root) are the seed blocks created in the first phase of duplication, while nodes at depth two (i.e. children of seed blocks) are the secondary duplication blocks constructed from substrings of one seed block and of the ancestral genome. (See Figure 3.3b.)*

For a given pair of ancestral and present-day genomes, a most-parsimonious two-step duplication tree is that which defines a partition of the duplication blocks into seed duplication blocks and secondary duplication blocks and defines the ancestral relationships between seed and secondary blocks such that the total number of duplication events needed to construct first the seed blocks and then the secondary blocks is minimum.

The total duplication distance for a two-step duplication tree is the sum of the number of duplicate operations needed to build all the duplication blocks. We express the number of duplicate operations needed to build a seed block B_i from the ancestral genome $\mathcal{G}$ as $d(\mathcal{G}, B_i)$. Secondary duplication blocks are built from substrings of both its parent seed block and the ancestral genome. Thus, we express the number of duplicate operations needed to build a secondary block B_j from its parent seed block B_i and $\mathcal{G}$ as $d(B_i \circ \mathcal{G}, B_j)$,

where $B_i \circ \mathcal{G}$ denotes the concatenation[1] of the strings B_i and $\mathcal{G}$.

We now have the following definition.

Definition 16. *Given the ancestral genome $\mathcal{G}$ and a set duplication blocks $B_1, \ldots, B_N$ from the present-day genome, a **most-parsimonious two-step duplication tree** is a two-step duplication tree (Def. 15) with minimum total duplication distance on its edges.*

We note that the definition of a most-parsimonious two-step duplication tree can be extended to more general distance measures. For example, a suitable measure of parsimony could be duplication-inversion distance or any of the other extensions of duplication distance presented in Chapter 2.

Now we show how to formulate the problem of constructing a most-parsimonious two-step duplication tree as an integer linear program (ILP).

A two-step duplication tree for a given ancestral genome and a set of duplication blocks is defined by a labeling of each of the N duplication blocks as either seed blocks or as secondary blocks. In addition to this labeling, we must also define for each secondary block which seed duplication block seeded it, i.e. which seed block is its parent in the tree.

A most-parsimonious two-step duplication tree is a solution of the following integer linear program.

$$\min_{U,V} \left[\sum_{i=1}^{N} (u_i \times d(\mathcal{G}, B_i)) + \sum_{i=1}^{N} \sum_{j=1}^{N} (v_{ij} \times d(B_j \circ \mathcal{G}, B_i)) \right] \tag{3.1}$$

such that

$$\sum_{j} v_{ij} = 1 \text{ for all } i \tag{3.2}$$

$$v_{ij} - u_j \leq 0 \text{ for all } i, j \tag{3.3}$$

$$u_i \in \{0, 1\} \text{ and } v_{ij} \in \{0, 1\}. \tag{3.4}$$

The binary variables $U = [u_1, \ldots, u_N]$ and binary matrix $V = [v_{ij}]_{i,j=1}^{N}$ describe the topology of the duplication tree. The binary variable u_i indicates whether a duplication block B_i is labeled as a seed block and thus defines an edge in the tree from the root $\mathcal{G}$ to B_i. The binary variable v_{ij} indicates that secondary duplication block B_i is seeded by seed block

[1] We insert a "dummy character" between B_i and $\mathcal{G}$ in the concatenate to avoid copying substrings across the boundary.

B_j and thus block B_i is a child of B_j in the tree. Again, we note that the duplication distance function, d, in the above program could be substituted by any other suitable distance function between strings with duplications, such as duplication-inversion distance.

We note that this program is equivalent to a special case of the facility location problem, a classic NP-hard combinatorial optimization problem. The input to the facility location problem is a set of customers and a set of potential facility sites. For each site, there is a cost associated with opening a facility, and for each site-customer pair, there is a cost associated with supplying that customer from a facility at that site. The objective is to minimize the total cost of opening facilities and supplying customers such that every customer is supplied by exactly one open facility. In the context of the two-step duplication tree, each duplication block is both a customer to be supplied and the site of a potential facility. Opening a facility at site B_i corresponds to classifying B_i as a seed duplication block and the cost of opening a facility corresponds to the cost of constructing that seed block by aggregating singleton duplicons from their ancestral loci. Supplying customer B_j from facility B_i corresponds to classifying B_j as a secondary block that is constructed from substrings of seed block B_i and the ancestral genome $\mathcal{G}$, and the cost of supplying B_j from B_i is equal to the duplication distance from B_j to B_i.

We implemented our two-step duplication tree method to analyze the ancestry of segmental duplications in the human genome using duplication-inversion distance as the measure of parsimony. We used data from [32] who identified 417 contiguous duplication blocks in the human genome (hg17, May 2004). The duplication blocks were comprised of mosaic patterns of a total of 4,692 distinct duplicon sequences. [32] delimited regions of homology for each duplicon, respectively, within the set of duplication blocks with some duplication blocks containing tens of thousands of duplicons. Then the authors partitioned the duplication blocks into 24 "clades" or groups that they believed to have been derived from a common seed block ancestor. The clade analysis done by [32] was based on a hierarchical clustering of the duplication blocks by comparing their respective duplicon contents without regard to the order or orientation of subsequences of duplicons within blocks.

To begin our analysis, we represented each of the 417 duplication block as a signed string on the alphabet of integers between -4692 and $+4692$.[2] We represented the ancestral genome G, containing a unique copy of each of the non-homologous ancestral duplicons,

[2] A total of 437 duplication blocks were identified in the study by [32] but 20 of these blocks were missing their duplicon annotations.

as the non-ambiguous string of all duplicons (with positive orientations) with dummy characters inserted in between every pair of characters, i.e. $G = +1 \odot +2 \odot \cdots \odot +4692$, where $\odot$ denotes a dummy character.

For each ordered pair of duplication blocks B_i, B_j, we computed the duplication-inversion distance $\bar{d}(B_i \circ \mathcal{G}, B_j)$ using the algorithm presented in Chapter 2 (Eq. 2.6). Note that, for every duplication block B_i, the distance $\bar{d}(\mathcal{G}, B_i)$ from $\mathcal{G}$ to that block, is equal to the length of the target block $\mid B_i \mid$. With these distances, we solved the ILP in Eq. 3.1 using the optimization package CPLEX. Given a solution to the ILP, for every index i such that the binary variable $u_i = 1$ (i.e. such that facility i is opened), we labeled the block B_i as a seed block. We labeled the remaining blocks as secondary blocks. For any pair of blocks B_i, B_j, if the binary variable $v_{ij} = 1$ in the solution, we designated secondary block B_i to be a child of seed block B_j (i.e. customer i is supplied from facility j). The resulting two-step duplication tree is shown in Figure 3.4.[3]

The two-step duplication tree for the 417 human duplication blocks exhibits 64 seed blocks with varying numbers of secondary blocks, respectively, ranging from 1 to 28. We compared our analysis to the clade analysis of [32]. Note that the clade analysis represents a partition of the duplication blocks into groups that are believed to have evolved from a common seeding "ancestor" block. Similarly, our two-step duplication tree defines groups of blocks that might have evolved from a common ancestor block, namely the groups of blocks defined by a subtree rooted at a given seed block. Furthermore, our two-step duplication tree defines putative ancestral relationships between duplication blocks, indicating which duplication blocks may have seeded others.

After computing our tree, we colored the nodes of the tree according to the clade partition computed by [32] in a post-process. A visual inspection of the two-step duplication tree reveals an interesting relationship between our analysis and that of [32]: many of the subtrees rooted at a seed block, called *seed block groups*, are monochromatic or nearly so. (The largest seed block groups can be viewed in Fig. 3.5.) This concordance between our analysis and that of [32] indicates that we discovered many of the same relationships between groups of duplication blocks. However, many of our seed block groups were not monochromatic. To quantify the agreeance of the two analyses, we computed a χ^2 test of independence for the set of seed block groups we derived and the set of clades derived by

[3] A previous version of this result appeared in [37]. Here we present a more recent, previously unpublished result. The difference between the previously published tree and that shown here owes to a revised annotation of the set of duplication blocks identified by [32].

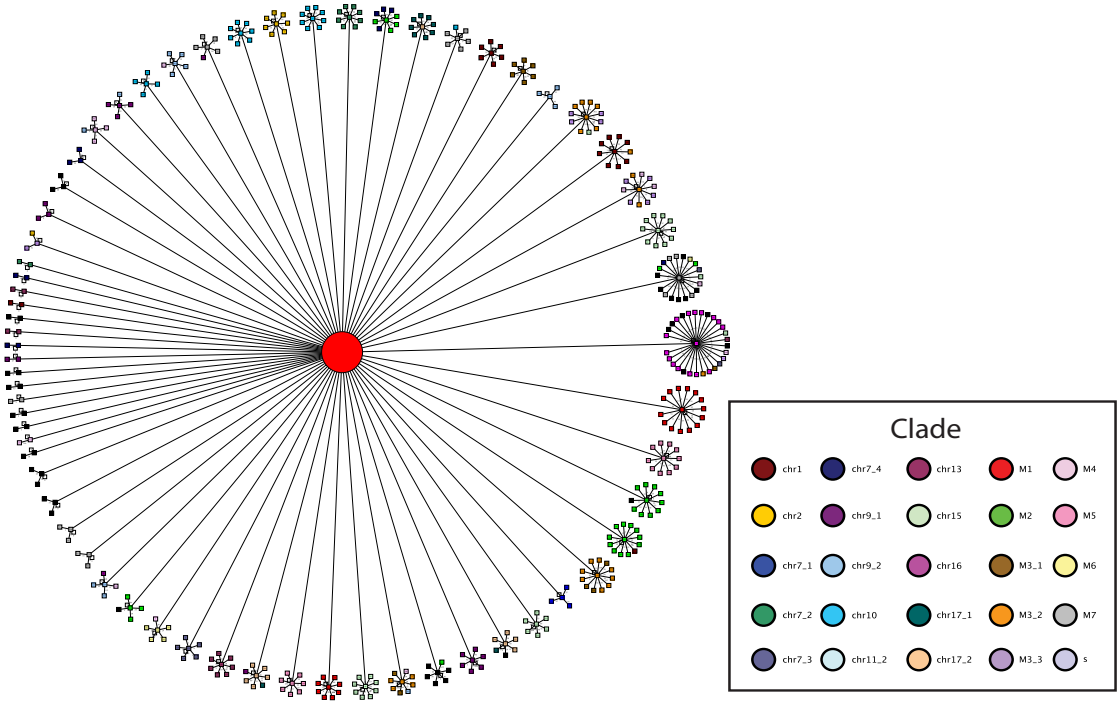

Figure 3.4: Most-parsimonious two-step duplication tree. The large, red node in the center is the root and represents the ancestral genome. The 64 children of the root are seed duplication blocks that were constructed during the first "seeding" phase of duplication. Children of seed blocks were constructed during the second phase of duplication. For comparison, the duplication blocks are colored according to the clade annotations computed in [32]. (Blocks labeled with clade 's' are members of small clades with five or fewer members.) The seed blocks are: chr9:94148625-94280597, chr2:86972987-87534573, chr2:95487038-95584611, chr13:51949404-52115934, chr2:131224724-131311234, chr17:21390703-21507201, chr6:58245619-58614492, chrY:7321346-7583561, chr20:23590986-23799725, chr19:142690-301857, chr9:41693541-41917754, chr13:22383730-22442581, chr18:14925636-15381131, chr2:111008489-111108421, chr9:67819771-68015022, chr7:22302593-22353864, chr15:21883445-22374842, chr7:64400020-64811464, chr21:32722334-32748407, chr22:21973667-22071991, chr17:16500312-16755448, chr2:106442349-106590011, chr22:23318989-23413657, chr2:131763427-131992854, chr6:5001-107014, chr19:59919900-60071479, chr7:45538339-45670718, chr15:30232700-30686935, chr15:28156284-28697532, chr1:13071685-13302468, chr15:28722450-28924396, chr15:75938308-76080230, chr18:14115138-14897871, chrY:2951337-3780270, chr16:21261363-21477613, chr13:92057461-92135789, chr12:36142962-36276062, chr5:174269510-174290127, chr2:97093305-97342494, chr7:63160249-63206018, chr17:23079959-23123088, chr3:126882689-127197788, chr17:4963474-5027382, chr2:94748046-95035488, chr5:49692886-49892733, chr15:26189051-26378746, chr1:561232-873944, chr2:97484061-97718125, chr16:18074902-18712195, chr15:76779895-76886440, chr21:13291342-14363850, chr7:56385628-56445155, chr1:145513740-145734052, chr7:56461815-56554511, chr21:28138407-28357448, chrY:12885909-13066979, chr7:43774170-43854733, chr22:15694297-15767503, chr1:142607398-142875550, chr20:23905387-23924629, chr7:55505324-55618047, chr7:63313785-63361829, chr7:57180803-57309010, chr1:142299774-142408068.

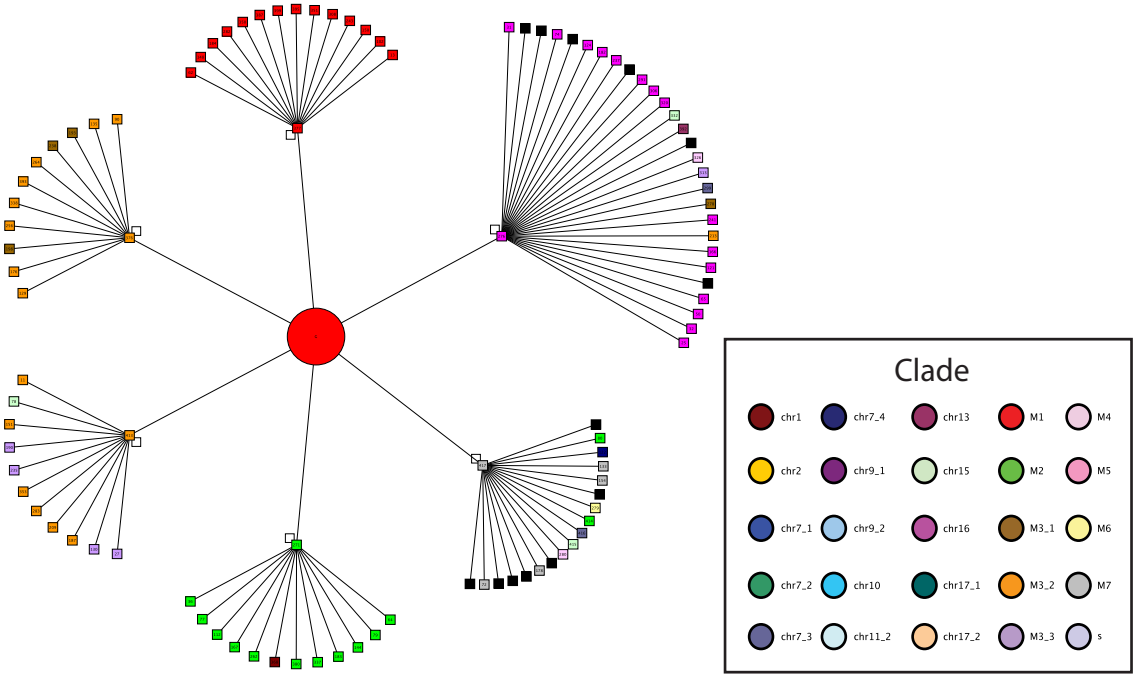

Figure 3.5: Most-parsimonious two-step duplication tree showing only subtrees of size at least 12. The large, red node in the center is the root and represents the ancestral genome. The 6 children of the root are seed duplication blocks that were constructed during the first "seeding" phase of duplication. Children of seed blocks were constructed during the second phase of duplication. For comparison, the duplication blocks are colored according to the clade annotations computed in [32]. The seed blocks are: chr2:106442349-106590011, chr9:41693541-41917754, chr12:8206636-8492606, chr7:22302593-22353864, chr15:26189051-26378746, chr18:14925636-15381131.

[32]. For groups containing at least six members (of which 34 seed block groups qualified and all 24 clades qualified), the probability that the correlation between the two categories was due to chance was $P < 0.35$.

Without strong evidence by which to conclude that the analysis of [32] corroborates ours, we sought to refine our analysis. Admittedly, our two-step duplication tree formulation interprets the two-step model of segmental duplication rather literally. It is unlikely that all the duplication blocks that exist in the human genome today were constructed in exactly two phases of duplication; it seems more plausible that perhaps several rounds of duplication took place with secondary blocks seeding tertiary blocks and tertiary blocks seeding quaternary blocks, etc. Moreover, if the model of duplication block construction via multiple rounds of duplication is plausible, then it seems reasonable to assume that any particular duplication block might have been constructed from multiple duplication events in which the duplicated material originated from more than one external seed block. That

is, the tree structure of our two-step duplication tree solution may be overly restrictive as some duplication blocks might have been seeded from multiple "parent" blocks.

In the next section, we refine our analysis of human duplication blocks by reformulating the problem of constructing a duplication history as the problem of computing an optimal directed acyclic graph (DAG) on the set of duplication blocks according to either a parsimony or a likelihood criterion.

3.2 The Max Parsimony and Max Likelihood Duplication History DAGs

Here we present a novel formulation of the problem of computing an evolutionary history for a set of segmental duplications that are organized in duplication blocks. We represent evolutionary relationships between a set of duplication blocks as a directed acyclic graph (DAG), relaxing some of the constraints of the problem formulation given in the previous section. We formalize the evolutionary reconstruction problem as an optimization over the space of DAGs.

We present two different methods for scoring a DAG: one based on parsimony and one based on likelihood. The parsimony score for a DAG is a straightforward extension of duplication distance that describes the most-parsimonious sequence of duplicate operations needed to construct a given target string. Because we have presented it in Chapter 2, here we forgo a description of the duplication distance measure or algorithm for computing it. The likelihood score for a DAG is the product of the likelihood scores for each of the duplication blocks, where a duplication block's likelihood is derived by computing the weighted ensemble of all possible duplication scenarios that could have generated it. We describe how to compute the partition function of the ensemble efficiently using a dynamic program that generalizes the duplication distance (i.e. parsimony score) recurrence. Deriving a probabilistic model from a dynamic program this way is analogous to the approach of [44] who applied dynamic programming to RNA folding to compute the partition function of all secondary structures and to assign probabilities to certain substructures..

Finally, we solve these evolutionary reconstruction problems on the set of duplication blocks identified by [32] using a local search technique based on simulated annealing. We compare these reconstructions to the analysis of [32]. Our evolutionary reconstruction recapitulates some of the properties of earlier analysis but also reveals additional and more subtle relationships between segmental duplications.

$$
\begin{aligned}
X &= && abcde \\
Y^0 &= && \emptyset \\
Y^1 &= Y^0 \circ \delta_X(1,3,1) = && abc \\
Y^2 &= Y^1 \circ \delta_X(4,5,1) = && deabc \\
Y &= Y^2 \circ \delta_X(4,5,5) = && deabdec
\end{aligned}
$$

Figure 3.6: An example of a sequence of duplicate operations that constructs $Y = deabdec$ from $X = abcde$. The corresponding feasible generator is: $\Psi_X = (X_{4,5}, X_{1,3}, X_{4,5}) = ((de), (abc), (de))$.

3.2.1 The Partition Function

We begin our discussion of the likelihood-based optimization problem with some preliminaries. Recall from Chapter 2 the following. Given a source/target pair X, Y, any sequence of duplicate operations of the form $\delta_X(s_1, t_1, p_1), \ldots, \delta_X(s_d, t_d, p_d)$ that generates Y from X uniquely partitions the characters of Y into non-overlapping subsequences corresponding to characters that were copied conjointly from X.

Definition 17. *Given a source string X, a **generator** $\Psi_X = (X_{i_1,j_1}, \ldots, X_{i_k,j_k})$ is a sequence of substrings of X.*

Definition 18. *A generator $\Psi_X = (X_{i_1,j_1}, \ldots, X_{i_k,j_k})$ is **feasible** for a target string Y, that we denote as $\Psi_X \dashv Y$, if:*

1. *The elements of Ψ_X partition the characters of Y into mutually non-overlapping subsequences $\{S_1, \ldots, S_k\}$.*
2. *There exists a bijective mapping $f : \{X_{i,j} \in \Psi_X\} \rightarrow \{S_1, \ldots, S_k\}$ from substrings of X to subsequences in Y corresponding to how the elements of Ψ_X partition Y.*
3. *The order of elements in Ψ_X corresponds to the order of the leftmost characters of the subsequences $f(X_{i_1,j_1}), \ldots, f(X_{i_k,j_k})$ in Y.*

See Fig. 3.6.

A sequence of k duplicate operations that constructs Y from X uniquely defines a feasible generator Ψ_X with length k whose elements correspond, respectively, to substrings of X that are duplicated conjointly in a single operation.

While a parsimony assumption is attractive from a theoretical perspective and can produce useful biological insight, it might be overly restrictive, particularly when there are many

different optimal or nearly optimal solutions. Consider, for example, the strings $X = abcdefghijkl$ and $Y = \mathbf{a}gd\mathbf{b}he\mathbf{c}if\mathbf{d}aj\mathbf{e}bk\mathbf{f}cl\mathbf{g}$. The duplication distance, $d(X, Y)$, is 13 and there is a single feasible generator with this optimum length. However, there are 989 possible feasible generators for Y, 119 of which have length 14, just slightly suboptimal. Because the space of all possible feasible generators is very large, a probabilistic model might give very low probability to an optimal parsimony solution. Thus, here we present a probabilistic model of segmental duplication that considers the weighted ensemble of all feasible generators for a source/target string pair.

For a given source string X and positive integer k we consider the space of all length-k generators Ψ_X. We define a probability distribution on the collection of generators by defining $Pr[\Psi_X] \propto \omega(\Psi_X)$ where $\omega(\Psi_X)$ is the "score," or weight, assigned to a generator, and we compute the partition function $Z_X^{(k)}$ of the weighted ensemble of all possible length-k generators Ψ_X. Given a source string X and a target string Y, we define the event F to be the event of choosing a length-k generator that is feasible for Y from the space of length-k generators. We define a probabilistic model for segmental duplications that, given a target string Y, assigns a probability to F: $Pr[F \mid Y, X, k]$. For a *fixed target string* Y, the probability, $Pr[F \mid Y, X, k]$, is the weighted ensemble of all possible length-k generators that are feasible for Y, normalized by the partition function $Z_X^{(k)}$. In particular, we can express the probability as:

$$Pr[F \mid Y, X, k] = \frac{1}{Z_X^{(k)}} \sum_{\Psi_X \dashv Y : |\Psi_X| = k} \omega(\Psi_X), \tag{3.5}$$

where $\mid \Psi_X \mid$ denotes the length of the generator. The likelihood of a target string Y, then can be expressed as $L(Y \mid F, X, k) = Pr(F \mid Y, X, k)$.

The score of a generator, $\omega(\Psi_X)$, can be defined according to various biological models. Although different functions ω may require different algorithms for computing the value $Pr[F \mid Y, X, k]$, we found that functions of the form $\omega(\Psi_X) = \sigma(\mid \Psi_X \mid, l(\Psi_X))$ where $l(\Psi_X) = \sum_{X_{i,j} \in \Psi_X} \mid X_{i,j} \mid$ denotes the sum of the lengths of the elements of Ψ_X, admit particularly efficient algorithms for computing Eq. (3.5). We discuss the score function further in Sec. 3.2.2.

Now we give an algorithm to compute the partition function, $Z_X^{(k)}$. Given a score function of the form $\sigma(\mid \Psi_X \mid, l(\Psi_X))$, each length-$k$ generator whose elements have lengths that sum to l are scored the same, namely $\sigma(k, l)$. Therefore, in order to compute $Z_X^{(k)}$, we must

calculate the total number of length-k generators whose lengths sum to l for all relevant values of l. Let $\mathcal{C}_X^{(k)}(l)$ equal the number of distinct length-k generators for which the sum of the lengths of the elements equals l.[4]

Lemma 5. *Let $X = x_1 \dots x_{|X|}$ be a source string and let k and l be positive integers. The function $\mathcal{C}_X^{(k)}(l)$ satisfies the following recurrence.*

$$\begin{aligned}\mathcal{C}_X^{(1)}(l) &= \mid X \mid -l+1, \\ \mathcal{C}_X^{(k)}(l) &= \sum_{l'=l-|X|}^{l-1} \mathcal{C}_X^{(k-1)}(l') \cdot (\mid X \mid -(l-l')+1).\end{aligned}$$

Proof: (Sketch) The base case, $\mathcal{C}_X^{(1)}(l)$, counts the number of distinct substrings of X with length l. In the case that $k = 1$, the number of substrings of X that have length l is equal to $X - l + 1$. Then, the value $\mathcal{C}_X^{(k)}(l)$ can be computed recursively by summing over all the ways of adding a k^{th} element to a set of $k-1$ substrings such that the resulting set of k elements has total length l. For a set of $k-1$ substrings of X whose lengths sum to l', there are $X - (l - l') + 1$ substrings of X that could be added to the set to yield a set of k substrings whose lengths sum to l.

For a source string X and integers k, l, if we are given $\mathcal{C}_X^{(k)}(l)$, we can compute $Z_X^{(k)}$ efficiently by summing $\mathcal{C}_X^{(k)}(l)$ over all relevant lengths l, weighting each feasible generator appropriately according to the function $\sigma(k, l)$. This gives the following theorem.

Theorem 5. *Let $X = x_1 \dots x_{|X|}$ be a source string and k be a positive integer. The partition function $Z_X^{(k)}$ satisfies the following.*

$$Z_X^{(k)} = \sum_{l=k}^{|X|\cdot k} \mathcal{C}_X^{(k)}(l) \cdot \sigma(k, l).$$

Note that the elements of a length-k list of substrings of X can have lengths that sum to at least k and at most $\mid X \mid \cdot k$.

The recurrence in Lemma 5 can be computed in $O(\mid X \mid k)$ time, so $Z_X^{(k)}$ can be computed in $O(\mid X \mid^2 k^2)$ time according to Theorem 5.

[4]The value $\mathcal{C}_X^{(k)}(l)$ is related to the well-known integer partition function $p(n)$ and corresponding Young tableaux. If $\mathcal{P}(l, k)$ is the set of partitions of the integer l into k parts, we can express $\mathcal{C}_X^{(k)}(l) = \sum_{P \in \mathcal{P}(l,k)} \sum_{\mathrm{p} \in P} (\mid X \mid -p+1) \cdot k!$.

3.2.2 The Score Function

We define the score of a generator $\omega(\Psi_X)$ to be some function that reflects the biological plausibility of the event of choosing a particular generator Ψ_X from the space of all generators and then duplicating the substrings of Ψ_X in some duplication scenario. When inferring a sequence of duplicate operations that can account for the construction of a particular target string Y by copying substrings of a particular source string X, a reasonable assumption is that the "simplest" explanation is the best. We consider the most-parsimonious duplication scenario–that is, the one requiring the fewest number of duplicate operations– to be the simplest. As noted above, the most-parsimonious solution can be computed using the duplication distance algorithm presented in [36, 37]. Therefore, our first consideration for scoring a generator is that one with small length, e.g. with length equal to the duplication distance, ought to have a good score.

Given a source string X, and two different target strings Y_1 and Y_2, where $\mid Y_1 \mid < \mid Y_2 \mid$, we assume that if the character contents of Y_1 and Y_2 are similar, then the construction of Y_1 from X is more likely than the construction of Y_2 from X. Again, this assumption favors simplicity. Therefore, two generators with length k that are feasible for Y_1 and Y_2, respectively, should be scored in a way that the generator for Y_1 is preferable to that for Y_2.

Theorems 2.7 and 2.9 allow for a score function of the form $\sigma(\mid \Psi_X \mid, l(\Psi_X))$. However, we impose two additional conditions that are biologically plausible.

a. For integers $k_1 < k_2$, $\sigma(k_1, l(\Psi_X)) > \sigma(k_2, l(\Psi_X))$. This property matches our intuition that a feasible generator with lesser cardinality (corresponding to a shorter sequence of duplicate operations needed to construct the target string) be more likely than a feasible generator with higher cardinality.

b. For identical source and target strings, $X = Y$, of length $\mid X \mid = \mid Y \mid > 1$, $\sigma(1, \mid Y \mid) > \sum_{\Psi_X : \mid \Psi_X \mid = k} \sigma(k, \mid Y \mid)$ for any $k > 1$. This will ensure that the event F of choosing a length-k generator for any $k > 1$ is less probable than choosing a length-1 generator; i.e. $Pr[F \mid Y, X, k] < Pr[F \mid Y, X, 1]$. This matches our intuition that the unique feasible generator of length 1 corresponding to the construction of Y by simply duplicating all of X in a single operation, will have higher probability than the combination of all feasible generators of length $k > 1$. Note that when $X \neq Y$, this property also ensures an analogous preference of feasible generators containing any long, contiguous substring $X_{s,t}$ that appears as a substring of Y over feasible

generators that contain fragmented portions of $X_{s,t}$ to generate the same substring in Y, all other elements being equal.

A suitable score function that meets these criteria is:

$$\sigma(k, \mid Y \mid) = \frac{1}{\mid Y \mid^k}. \tag{3.6}$$

Undoubtedly, there are other biologically motivated score functions that may produce meaningful results.

3.2.3 Restricted Partition Function

In this section, we present the final ingredient necessary to compute the probability $Pr[F \mid Y, X, k]$, namely the sum in Eq. (3.5) that we define as $Q_X^{(k)}(Y)$. We refer to the value $Q_X^{(k)}(Y)$ as the *restricted partition function of feasible generators*, and it is equal to the weighted ensemble of all length-k generators Ψ_X that are feasible for Y. Hence $Q_X^{(k)}(Y) = \sum_{\Psi_X \dashv Y : |\Psi_X| = k} \omega(\Psi_X) = \sum_{\Psi_X \dashv Y : |\Psi_X| = k} \sigma(k, \mid Y \mid)$.

In order to compute this value, we generalize the recurrence presented in Chapter 2 for computing duplication distance from source string X to target string Y to count the number of length-k generators that are feasible for Y.

Lemma 6. *Given a source string $X = x_1 \dots x_{|X|}$ and a target string $Y = y_1, \dots, y_{|Y|}$, the number $N_X^{(k)}(Y)$ of distinct length-k generators Ψ_X that are feasible for Y satisfies the following recurrence.*

$$\begin{aligned}
N_X^{(k)}(Y) &= \sum_{i : x_i = y_1} N_X^{(k)}(Y, i), \\
N_X^{(1)}(Y, i) &= \begin{cases} 1 & \text{if } Y = X_{i, i+|Y|-1}, \\ 0 & \text{otherwise,} \end{cases} \\
N_X^{(k)}(Y, i) &= N_X^{(k-1)}(Y_{2,|Y|}) + \sum_{j > 1 : y_j = x_{i+1}} \sum_{l=1}^{k} [N_X^{(l)}(Y_{2,j-1}) \cdot N_X^{(k-l)}(Y_{j,|Y|}, i+1)].
\end{aligned}$$

Here, the term $N_X^{(k)}(Y, i)$ represents the number of feasible generators Ψ_X with length k given that the character y_1 is generated by a substring of X starting at x_i.

First, we give intuition for the recurrence in Lemma 6, and then we sketch a proof of its correctness. We note that the proof of correctness for Lemma 6 mirrors, in many ways,

the proof of correctness of Theorem 1 that gives a recurrence for computing a minimum-length feasible generator for a source string X and a target string Y; here, instead, we want to count the total number of feasible generators Ψ_X that have a fixed length k.

Recall the non-overlapping property stated as Lemma 1. The recurrence for computing $N_X^{(k)}(Y)$ is efficient because the non-overlapping property allows us to subdivide the characters of the target string Y into independent subproblems. For example, if we are considering the set of feasible generators that contain some subsequence S of Y, $\mathcal{S}_S = \{\Psi_X : S \in \Psi_X\}$, then for every $\Psi_X \in \mathcal{S}_S$, all other elements of Ψ_X cannot overlap the characters in S. Therefore, the substrings of Y in between successive characters of S define subproblems that can be computed independently.

Note that every character of Y must appear at least once in X. In order to count the number of feasible generators Ψ_X with length $k > 1$, we must consider all subsequences of Y that could have been generated by a single duplicate operation and the number of ways we could combine exactly k of those subsequences to form a feasible generator Ψ_X. The recurrence is based on the observation that in any feasible generator, Ψ_X, y_1 must be the first (i.e. leftmost) character in some element of Ψ_X. There are then two cases to consider: either (1) y_1 was the last (or rightmost) character in the substring that was duplicated from X to generate y_1, or (2) y_1 was not the last character in the substring that was duplicated from X to generate y_1.

Proof: (sketch)

The recurrence defines two quantities: $N_X^{(k)}(Y)$ and $N_X^{(k)}(Y, i)$. We shall show, by induction, on $\mid Y \mid$ and k that for a pair of strings, X and Y, the value $N_X^{(k)}(Y)$ is equal to the number of length-k feasible generators Ψ_X, and that $N_X^{(k)}(Y, i)$ is equal to the number of length-k feasible generators Ψ_X under the restriction that the character y_1 is copied from index i in X, i.e. x_i *generates* y_1. $N_X^{(k)}(Y)$ is computed by summing over all characters x_i of X that can generate y_1..

As described above, we must consider two possibilities in order to compute $N_X^{(k)}(Y)$. In every feasible generator Ψ_X, the character y_1 must appear in some subsequence $S_{y_1} \in \Psi_X$ of Y that contains y_1 as a leftmost character and that corresponds to a substring of X that was copied conjointly to produce the subsequence S_{y_1}. Either:

- Case 1: y_1 was the last (or rightmost) character in the substring of X that was copied to produce y_1, i.e. S_{y_1} has length 1, or

- Case 2: x_{i+1} is also copied in the same duplicate operation as x_i, possibly along with other characters as well, i.e. S_{y_1} has length greater than 1.

For case one, number of length-k feasible generators Ψ_X is equal to the number of length-$(k-1)$ feasible generators $\Psi_X(Y_{2,|Y|})$ for source string X and target string $Y_{2,|Y|}$ (the suffix of Y); the union of the subsequence corresponding to the single character y_1 and any length-$(k-1)$ feasible generator $\Psi_X(Y_{2,|Y|})$ results in a length-k feasible generator Ψ_X. For case two, Lemma 1 implies that the total number of length-k feasible generators Ψ_X is the product of two independent subproblems. Specifically, for each $j > 1$ such that $x_{i+1} = y_j$ and for each $l \in \{1, 2, \ldots, k\}$, we compute: (i) number of length-l feasible generators for source string X and target string $Y_{2,j-1}$, namely $N_X^{(l)}(Y_{2,j-1})$, and (ii) the number of length-$(k-l)$ feasible generators for source string X and target string $y_1 Y_{j,|Y|}$ that include an element S_{y_1} in which y_1 is generated by x_i. To compute the latter, recall that all relevant feasible generators (corresponding to case 2 above) Ψ_X must contain an element that corresponds to a duplicate operation in which x_i and x_{i+1} are copied conjointly. The number of relevant length-$(k-l)$ feasible generators for source string X and target string $y_1 Y_{j,|Y|}$ that contain an element S_{y_1} that corresponds to a substring of X starting at x_i and also containing x_{i+1} is equal to the number of relevant length-$(k-l)$ feasible generators for source string X and target string $Y_{j,|Y|}$ that contain some element S_{y_j} that corresponds to a substring of X starting at x_{i+1}, namely $N_X^{(k-1)}(Y_{j,|Y|}, i+1)$. ■

We compute the restricted partition function $Q_X^{(k)}(Y)$ efficiently by first counting the number of relevant feasible generators, namely $N_X^{(k)}(Y)$, and scoring each generator appropriately by $\sigma(k, \mid Y \mid)$. This gives us the following theorem.

Theorem 6. *Let $X = x_1 \ldots x_{|X|}$, $Y = y_1, \ldots, y_{|Y|}$ be a source/target string pair and let k be a positive integer. The restricted partition function $Q_X^{(k)}(Y)$ satisfies the following.*

$$Q_X^{(k)}(Y) = N_X^{(k)}(Y) \cdot \sigma(k, \mid Y \mid).$$

The recurrence given in Lemma 6 can be computed in time $O(\mid Y \mid^2 k^2 \mu(Y)\mu(X))$ where $\mu(Y)$ (resp. $\mu(X)$) is the maximum multiplicity of any character that appears in Y (resp. X), so computing $Q_X^{(k)}(Y)$ takes the same time.

The probabilistic model of duplication based on the ensemble of feasible generators presented in this chapter is just one of many possible models one could imagine. Ultimately, a model ought to reflect a reasonable approximation of a biologically plausible events, but

should also admit an efficiently computable algorithm. We give another generative probabilistic model of duplication in Appendix B.

3.2.4 Problem Formulation

Here we formalize the problem of computing a segmental duplication evolutionary history for a set of duplication blocks in the human genome with respect to either a parsimony or likelihood criterion.

The input to our problem is the set of duplication blocks found in the human genome, each represented as a signed string on the alphabet of duplicons. Our goal is to compute a putative duplication history that accounts for the construction of all of the duplication blocks. We assume that the ancestral genome is devoid of segmental duplications. A duplication history is a sequence of duplicate events that first builds up a set of *seed* duplication blocks by duplicating and aggregating duplicons from their ancestral loci and then successively constructs the remaining duplication blocks by duplicating substrings of previously constructed blocks.

We observed in [36] strong evidence that many of the duplication blocks identified by [32] had been constructed through the duplication and aggregation of substrings of duplicons from several other blocks. Therefore, a tree cannot aptly represent an evolutionary history; a more appropriate representation of the evolutionary relationships between duplication blocks is a DAG in which the vertices represent duplication blocks and an edge directed from a vertex X to a vertex Y indicates that substrings of X were duplicated in the construction of Y. A vertex with multiple incoming edges and, therefore, multiple parents, is constructed using substrings of all of the parent blocks. Specifically, given a DAG $G = (\mathcal{D}, E)$, for $Y \in \mathcal{D}$, we define $P_G(Y)$, the parent string of Y, by $P_G(Y) = X_1 \odot X_2 \odot \cdots \odot X_p$ where $X_i \in \{\mathcal{D} \mid (X_i, Y) \in E\}$ and $\odot$ indicates the concatenation of two strings with a dummy character inserted in between.

We make two simplifying assumptions. First, we assume that only duplicate events occur and that there are no deletions, inversions, or other types of rearrangements within a duplication block. Second, we assume that a duplication block is not copied and used to make another duplication block until *after* it has been fully constructed, ensuring the evolutionary relationships cannot contain cycles. We acknowledge that our two simplifying assumptions restrict the evolutionary history reconstruction problem significantly, but admit an efficient and consistent method of scoring a solution. Similar assumptions were made, for example,

by [52] to derive the evolutionary tree for Alu repeat elements.

We can define the optimal DAG with respect to a parsimony criterion using duplication distance (see Ch. 2).

Definition 19. *Given a set of duplication blocks $\mathcal{D}$, the **maximum parsimony evolutionary history** is the DAG $G = (\mathcal{D}, E)$ that minimizes $f(G) = \sum_{Y \in \mathcal{D}} d(P_G(Y), Y)$.*

We can also define the optimal DAG with respect to a likelihood criterion. In phylogenetic tree reconstruction, a max likelihood solution is a tree that maximizes the probability of generating the characters at the leaf nodes over all possible tree topologies, branch lengths, and assignments of ancestral states to the internal nodes. Typically, the evolutionary process is assumed to be a Markov process so that the probabilities along different branches are independent. We similarly define the maximum likelihood DAG using the probabilistic model derived in Section **??**. We maximize the likelihood of the solution over all DAG topologies and–instead of branch lengths–the numbers of operations permitted to construct each node.

Definition 20. *Given a set of duplication blocks $\mathcal{D}$, the **maximum likelihood evolutionary history** is the DAG $G = (\mathcal{D}, E)$ that maximizes the likelihood:*

$$\begin{aligned} L(G) &= \prod_{Y \in \mathcal{D}} L(Y), \\ &= \prod_{Y \in \mathcal{D}} \left(\max_k Pr[F \mid Y, P_G(Y), k] \right), \\ &= \prod_{Y \in \mathcal{D}} \left(\max_k Q^{(k)}_{P_G(Y)}(Y) / Z^{(k)}_{P_G(Y)} \right), \end{aligned}$$

where $Z^{(k)}_{P_G(X)}$ and $Q^{(k)}_{P_G(Y)}$ are the partition function and restricted partition functions, respectively.

3.2.5 Results

We analyzed a set of 391 duplication blocks identified by [32] that were represented as signed strings on an alphabet of $\approx 5,000$ duplicons. We computed the maximum parsimony evolutionary history (Def. 19) for the entire set of blocks (see Fig. 3.7). The DAG exhibited multiple connected components. For comparison, we then computed the maximum likelihood evolutionary histories (Def. 20) for several of the subgraphs induced by connected components of the parsimony solution. We scored generators according to $\sigma(k, \mid Y \mid) = \frac{1}{|Y|^k}$.

We used a simulated annealing strategy to find a maximum parsimony DAG for the entire

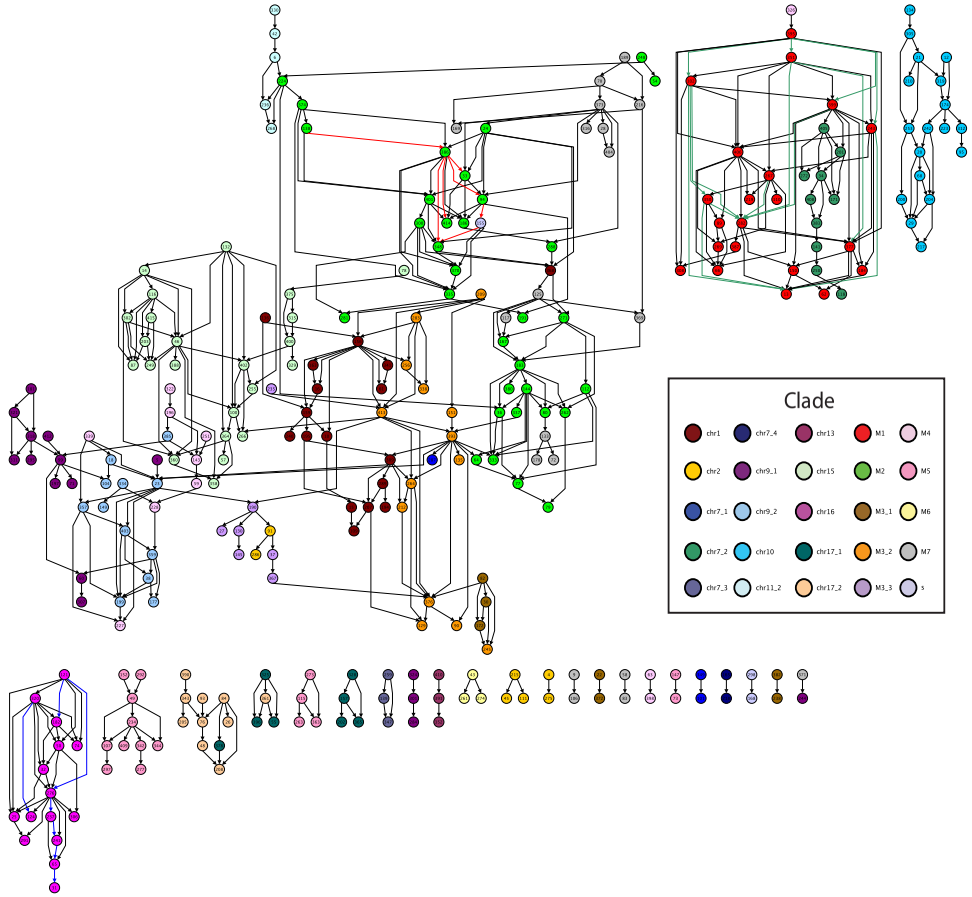

Figure 3.7: The maximum parsimony DAG for a set of 391 duplication blocks in the human genome. Edges indicate evolutionary relations; an edge is directed from a node u to a node v if the most-parsimonious duplication scenario includes duplication events that copy substrings of u in the construction of v. [32] partitioned the duplication blocks into a set of 24 clades (plus one 's' group of duplication blocks found in subtelomeric regions) that we indicate here with 25 colors on nodes. The 3 sets of colored edges represent inheritance networks for 3 conserved subsequences of duplicons. These inheritance networks are almost entirely confined to a single clade each. The green edges represent the inheritance of the duplicon sequence $[6968, 6967, 6965, 6963, 6962, 6960]$ in clade 'M1', the red edges represent the inheritance of $[7039, 7036, 7037]$ in clade 'M2', and the blue edges represent the inheritance of $[9448, 9449]$ in clade 'chr16.'

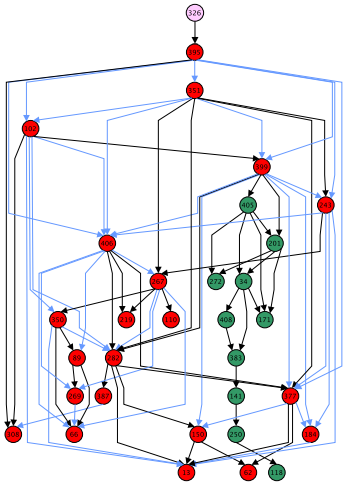

Figure 3.8: A connected component of the maximum parsimony DAG. Nodes from clade 'M1' are red and nodes from clade 'chr7_2' are green. Node labels correspond to duplication block IDs. The blue edges represents the inheritance network for non-core duplicon 6970.

set of duplication blocks and to find maximum likelihood DAGs for several subgraphs.[5] For each input, we ran our local search 300 times. We started the search an equal number of times at each of three different types of initial graphs: (a) the empty graph with no edges; (b) the directed minimum spanning tree (MST); and (c) a randomly chosen DAG (chosen independently for each trial). Finally, to focus the search on the most important block relationships, we considered only edges between blocks whose longest common subsequence (LCS) contained at least 20 duplicons. We describe the simulated annealing heuristic in more detail in Section 3.2.8.

3.2.6 Maximum Parsimony Reconstruction

The maximum parsimony DAG contains 391 nodes and 479 edges. There are 9 connected components with at least 4 duplication blocks, and nearly 40% of the blocks appear in the largest connected component. Figure 3.8 shows a moderately-sized connected component. The graph also contains a total of 105 singleton nodes for which we did not infer any evolutionary relations with other duplication blocks, 97 of which did not exhibit an LCS of length 20 with any other block.

The maximum parsimony DAG represents a scenario in which all 391 duplication blocks

[5]Both the max parsimony and max likelihood versions of the problem can be shown to be NP-hard by a reduction from the problem of Learning Bayesian Networks.

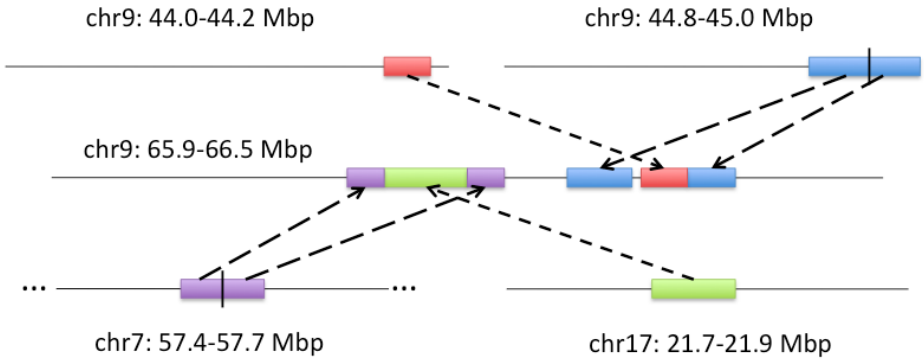

Figure 3.9: An example of duplication block recombination. The target duplication block in the middle (chr9:65.9-66.5) exhibits subsequences that appear as contiguous substrings in four other seeding duplication blocks. The nested relationships between subsequences in the target block (e.g. the green subsequence is nested inside the purple one and the red subsequence is nested inside the blue one) allow us to conclude the target block was composed of duplicated substrings from the other four blocks and not vice versa. Moreover, the nested relationships between these subsequences imply an order of duplication events (i.e. the green subsequence was duplicated after the purple one and the red subsequence was duplicated after the blue one).

could have been constructed in a sequence of 17,431 total duplicate operations. As a baseline comparison, a minimum spanning tree, with respect to duplication distance, on the set of duplication blocks has a total parsimony score of 28,852 and, by definition, contains 390 edges.

A striking feature of the max parsimony DAG was the occurrence of *duplication block recombination*, or the creation of a single target block by duplicating and aggregating substrings from multiple parent blocks. See Fig. 3.9 for an example of a duplication block that exhibits a subsequence composition that can best be described as the result of seeding events involving multiple parents. The existence of a duplication block that contains subsequences contributed by multiple parent blocks was not possible in the two-step duplication tree formulation presented in Section 3.1, underscoring the differences between the two approaches. In total, 128 duplication blocks exhibited multiple parents. Of those, 105 exhibited at least two parents that contributed, respectively, at least 10% of the duplicon content of the target node (computed as the number of duplicons in the subsequences contributed by a parent divided by the total number of duplicons in the target). Similarly, 66 blocks exhibited at least two parents that contributed, respectively, at least 20% of their constituent duplicons. And 52 exhibited at least two parents that contributed, respectively, at least 25% of the content of the target block.

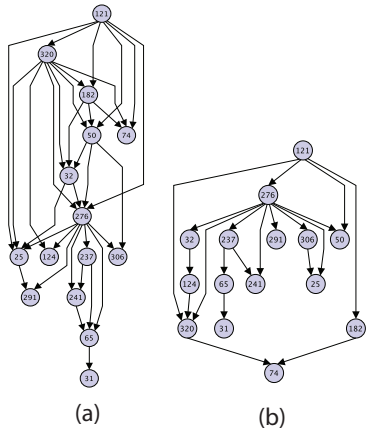

Figure 3.10: (a) Component comprised entirely of duplication blocks from clade 'chr16' in the maximum parsimony DAG. (b) Maximum likelihood DAG for subgraph induced on nodes in (a).

[32] performed an initial analysis of the duplication blocks. They defined 24 *clades*, or groups of duplication blocks derived from a common ancestor block, by performing hierarchical clustering on a matrix representing the shared presence or absence of duplicons for every pair of blocks. For a given clade they defined a *core duplicon* as one that appears in at least 67% of the constituent duplication blocks. They posited that clades represent families of evolutionarily related duplication blocks and that core duplicons "may have driven the evolution of the duplication blocks" in a clade.

After constructing the max parsimony DAG, we colored the nodes in a post-process according to the clades described in [32]. We found a strong correspondence between Jiang et al.'s clades and connected subgraphs in our DAG; 5 of the 9 connected components with at least 4 blocks were comprised of duplication blocks belonging to a single clade and 7 of the 9 components were comprised of blocks belonging to no more than 2 clades. For example, see Fig. 3.10(a) and 3.11(a). In larger components, nodes from a single clade frequently induce a connected subgraph. For example, see Fig. 3.8. We performed a χ^2 test of independence between the members of the clades defined by [32] and the members of connected components in our graph. We restricted the test to clade and components with at least 18 members. We found that there was a strong relation between the partition of our graph into connected components and the clade analysis done by [32] ($P < 0.0001$). The analysis done by [32], therefore, corroborates our own conclusions.

Our DAG also reveals which duplication blocks may have seeded many other blocks (i.e. those with high out-degree). For example, in Fig. 3.8, block 399 exhibits eight children and is an inflection point for the component. Moreover, the edge from block 399 to 405 links

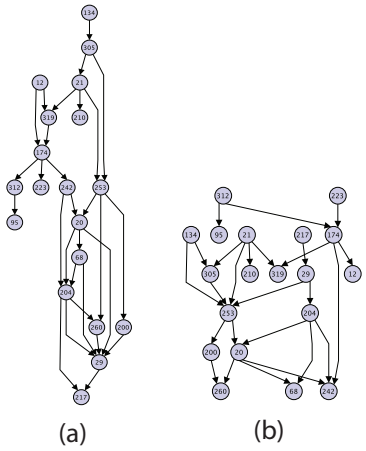

Figure 3.11: (a) Component comprised entirely of duplication blocks from clade 'chr10' in the maximum parsimony DAG. (b) Maximum likelihood DAG for subgraph induced on nodes in (a).

blocks from the the 'M1' and 'chr7_2' clades. Even though the blocks 399 and 405 belong to different clades, 405 is very "close" to 399 in duplication distance: block 405 contains only 71 duplicons, but it shares a subsequence of 29 duplicons with block 399. This link suggests that the entirety of clade 'chr7_2' was descended from clade 'M1' in an optimal duplication history.

Also implicit in the DAG is information about which duplicons are duplicated from one block to another in an optimal duplication history. We define the *inheritance network* for each duplicon as the subgraph induced on the edges on which that duplicon is passed from parent to child. The average size of an inheritance network was 5.5 edges with a standard deviation of 10.4. As expected, the 81 core duplicons identified by [32] were more promiscuous, on average, than non-core duplicons with a mean inheritance network size of 21. Interestingly, a comparison of the inheritance networks for core and non-core duplicons revealed that many non-core duplicons exhibit larger inheritance networks within subgraphs induced by a clade than many of the core duplicons. For example, non-core duplicon 6970 appeared on 36 of the 63 total edges in the subgraph induced by clade 'M1' (shown in blue in Fig. 3.8) and does not appear on any other edge in the graph. We propose 6970 as a new core duplicon for this clade and suggest that others like it should also be categorized as core duplicons.

Moreover, we found inheritance networks for many conserved subsequences of duplicons that were nearly as prominent as those for individual core duplicons. For example, the subsequence $[6968, 6967, 6925, 6963, 6962]$ of duplicons appears on 23 of the edges in the

subgraph induced by ‘M1’ clade nodes (shown as green edges in Fig. 3.7). Similarly, the sequence $[7039, 7036, 7037]$ exhibits a connected inheritance network of 7 edges within the subgraph induced on clade ‘M2,’ and $[9448, 9449]$ exhibits an inheritance network of 7 edges within the subgraph induced on clade ‘chr16’ that includes an inheritance path of length 5 (shown also in Fig. 3.7). By delineating the inheritance networks of duplicon subsequences that are conserved across duplication blocks, we can learn about which duplicons were duplicated and transposed conjointly. This type of analysis was impossible using only the clade annotations of [32].

3.2.7 Maximum Likelihood Reconstruction

We computed the maximum likelihood DAGs (Def. 20) for the sets of duplication blocks appearing within moderately-sized connected components of the maximum parsimony DAG in order to compare the two methods. We chose the components comprised of blocks from clades ‘chr16’ and ’chr10’, respectively (in Fig. 3.7). The maximum likelihood subgraphs for these subproblems are shown in Fig. 3.10(b) and 3.11(b).

The two DAGs for the ‘chr16’ component in Fig. 3.10 share some characteristics. For example, node 121 is a common ancestor of every other block and block 276 exhibits high out-degree in both solutions. Both solutions are similarly “good” with respect to the parsimony objective: the solution in (a) exhibits an optimal parsimony score of 397, and the one in (b) exhibits a score of 401. However, the likelihood score for the parsimony solution in (a) was nearly zero. One difference that accounts for this discrepancy is the higher average in-degree for blocks in the parsimony solution (2.2) as compared to the likelihood solution (1.3). Also, the parsimony solution exhibits a path with ten edges, whereas the longest path in the likelihood solution has six.

Some of these differences are due to the fact that the parsimony criterion does not penalize edges that do not directly improve the score. For example, block 291 has two parents (276 and 25) in the parsimony DAG but only one parent (276) in the likelihood DAG. However, the duplication distance with source $276 \odot 25$ and target 291 is the same as the duplication distance with source 276 and target 291. Therefore, the edge from 25 to 291 does not improve the parsimony score, underscoring that there are multiple optimal parsimony solutions. In contrast, the likelihood of a target block generally increases as the sum of the lengths of its parent blocks decreases, so the max likelihood DAG will not include edges that do not directly improve the score.

3.2.8 The Simulated Annealing Heuristic

[29] describes an elegant approach for moving locally in the space of DAGs via three types of simple moves- *adding* a new edge, *removing* an existing one, or *reversing* an existing one.

Definition 21. *Given DAG $G = (V, E)$, the DAG $G' = (V, E')$* ***neighbors*** *G if and only if we can obtain G' from G with a single move- adding a new edge, removing, or reversing an existing edge.*

Definition 22. *Given an objective function f and two DAGs G_1, G_2 we call $\Delta G = f(G_1) - f(G_2)$ the* ***difference*** *in their energies.*

The simulated annealing strategy can be summarized as follows. Given a DAG $G = (V, E)$ and a randomly proposed move, we examine whether the move is legal (i.e. does not induce a cycle) and, if so, we perform the move with probability $p = exp(\frac{-\Delta G}{T})$, where T is a temperature parameter. We note that depending on the complexity of the objective function $f(G)$ computing ΔG could be very expensive. In fact, this is the case for the max likelihood reconstruction because computing $Pr[Y \mid X, k]$ takes in the worst-case $O(\mid Y \mid^3 \mid X \mid k^2)$. Therefore, we employ a hashtable to store the cost of every move we have examined. As we do hundreds of independent trials we may often need to examine the same move multiple times, and the hashtable helps significantly speed up the search for good moves..

In our implementation we employ an exponential cooling schedule schedule. The temperature is updated via the equation $T_{t+1} = T_t\alpha$. We determined empirically that $\alpha = 0.98$ performs best in terms of efficiency and time.

The simulated annealing heuristic often terminated in local optima. For a particular instance, the solutions found by all 300 trials would include many globally suboptimal solutions. However, many of the locally optimal solutions encountered were "close" to the score for the best solution found. For example, the search for the max parsimony evolutionary history given in Fig. 3.10(a) resulted in a component whose objective score is 397; more than 1/6 of the total trials returned solutions whose objective scores are no more than 407 and well over 1/2 of the total trials returned solutions whose objective scores are no more than 437 (see Fig. 3.12).

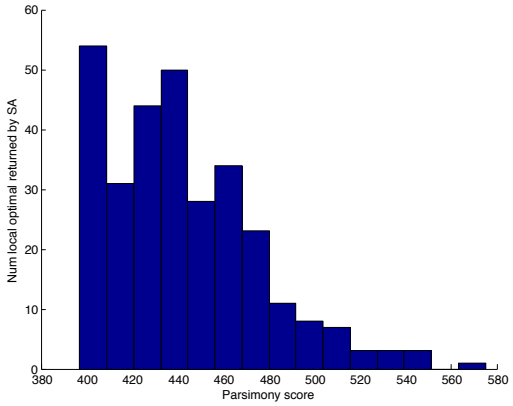

Figure 3.12: Results of 300 trials of simulated annealing (SA) heuristic: number of local optima returned by SA vs. objective scores. Results are from search for max parsimony evolutionary history for component comprised of duplication blocks from clade 'chr16' whose global optimum is given in Fig.3(a).

3.3 Conclusion and Future Directions

We have given several methods for constructing a putative history of human segmental duplications. First, we defined the problem of computing a most-parsimonious duplication history for segmental duplications that is consistent with the so-called *two-step model* of segmental duplication. We computed an optimal putative duplication history for a set of human segmental duplications using the duplication distance algorithm described in Chapter 2 and an integer linear program problem formulation. We then refined the segmental duplication history reconstruction problem by relaxing the constraint that the history must be consistent with the two-step model, and instead be consistent more generally with a multi-step model. We defined a probabilistic model of duplication and gave an efficient algorithm for computing the probability of a duplication scenario for a pair of signed strings by computing the partition function of the weighted ensemble of feasible sets of duplicated segments. Then we defined the problem of constructing an optimal segmental duplication history DAG with respect to both parsimony and likelihood criteria. Finally, we computed a near-optimal putative duplication history DAG for the same set of human segmental duplications with respect to the parsimony criterion and a near-optimal putative duplication history DAG for a subset of the same human segmental duplications with respect to the likelihood criterion using.

Our maximum parsimony and maximum likelihood DAG reconstructions show some differences, both from each other and from the analysis of [32]. In particular, we identify non-core duplicons and subsequences that are arguably as promiscuous within a clade as core duplicons.

There are several directions for future work. From a theoretical perspective, one can incorporate other types of operations into the probabilistic model, such as deletions and inversions (which we described in the parsimony setting in Chapter 2), as well as single nucleotide mutations. Also, our method could be used to sample over the space of DAGs using a Markov Chain Monte Carlo (MCMC) strategy. From the perspective of applications, a more comprehensive analysis of genes or other elements in the newly identified core duplicons and core subsequences from our reconstruction is warranted, as is a further refinement of the clade annotation by analyzing the clade-induced subgraphs of the DAGs.

COMPLETING A PARTIALLY ASSEMBLED GENOME USING REARRANGEMENT DISTANCE

The growing abundance of cancer genome sequence data (see [51], for example) underscores the need for efficient and reliable algorithms for whole-genome assembly. However, assembling a genome accurately is still a costly and labor-intensive process. The need to survey many tumor samples means that the sequencing and finishing efforts devoted to a single genome are likely to be relatively small so we need computational methods to aid the process.

Unfortunately, in order to keep the cost of sequencing a single cancer genome down, the sequence coverage may be low. The effect is that a cancer genome may not be able to be reconstructed fully from the sequence data. Instead, the reconstruction may consist of a fragmented genome comprised of multiple DNA contigs whose sequences are known but whose relative ordering cannot be inferred directly, and some regions may not have been measured at all. While this fragmented representation of a cancer genome can be used, for example, to identify regions of amplification or deletion, we cannot fully appreciate the effects of large-scale rearrangements without full cancer genome reconstructions.

In some cases, we have a good idea of what the full genome should look like. Unlike when sequencing a new species, in assembling a cancer genome, we know the architecture of the "starting genome," i.e. the human genome prior to somatic mutation. We assume, conservatively, that a human cancer genome will exhibit some mutations, but will, by and large, resemble a healthy human genome. Given a partially assembled cancer genome comprised of multiple fragmented contigs, we can reconcile it with a reference (healthy) genome to construct a full cancer genome that contains all the assembled contigs as subsequences but that is otherwise as similar to the reference genome as possible. In particular, where regions

may not have been measured in the cancer sequence data, we assume that the those regions resemble the reference genome. Moreover, whenever there is ambiguity in how to arrange the fragmented cancer genome contigs, we err on the side of conservatism and use the reference genome as our guide. This process, known as *resequencing* a cancer genome, requires fewer clones and is less labor-intensive than de novo whole genome fragment assembly.

In this chapter, we consider the *Block Ordering Problem (BOP)* that was introduced by Gaul and Blanchette [28]. The input to the BOP is a pair of partially assembled genomes where all the regions of the genomes are sequenced, but the genomes are fragmented into contigs (blocks) that need to be ordered and oriented such that the similarity between the resulting pair of genomes is maximized. In the formulation of [28], the measure of similarity between a pair of genomes is given by the number of cycles in their breakpoint graph ([5]), and they give a linear-time algorithm to solve it.

Here we consider a special case of the BOP where the input is one fully sequenced reference genome and one partially assembled genome where some regions may not have been sequenced. This special case is motivated by the problem of resequencing a cancer genome: given a fully sequenced reference genome and a partially assembled cancer genome comprised of multiple contigs possibly with some missing regions, complete the cancer genome in such a way that it contains all the assembled contigs as subsequences and such that the similarity between the cancer genome and the reference genome is maximized. We note, however, that the techniques we present can be extended in a straightforward manner to solve the general BOP in which both input genomes are only partially assembled.

Unlike [28], here we use the double-cut-and-join (DCJ) distance metric between genomes as a measure of similarity. First introduced by [59], DCJ distance is an efficiently computable metric that models a number of basic rearrangements, such as inversions, translocations, fusions, fissions, transpositions, and block interchanges. In [59], the authors present a linear-time algorithm for computing DCJ distance between a pair of genomes using the breakpoint graph data structure introduced by [5]. Later, [9] introduced a new data structure, the adjacency graph, that simplifies the algorithms for computing DCJ distance and for computing a sorting sequence of DCJ operations.

In this chapter, we show that solving the BOP with respect to DCJ distance is equivalent to solving it with respect to the number of cycles in the breakpoint graph; in fact, another contribution of this thesis is a proof that the number of cycles in the breakpoint graph for

a pair of genomes is equal to the number of cycles in their adjacency graph using cycle-preserving graph transformations. Moreover, just as [9] simplified the presentation of the algorithms for computing DCJ distance using an adjacency graph, here we simplify the presentation of the algorithm for solving the BOP by using the adjacency graph (instead of the breakpoint-graph-based framework used by [28]). We differentiate between two variants of the problem and give linear-time algorithms to solve them both. Finally, [28] do not give a full proof of correctness for their algorithm; we complete the proof of correctness for our algorithm.

In the last section of this chapter, we discuss how our adjacency-graph-based framework for solving the BOP might give insight into a more general problem of constructing a set of unknown cancer genomes given an ambiguous set of sequence measurements.

4.1 Related Work

As mentioned previously, the work in this chapter is closely related to that of Gaul and Blanchette [28] who introduced the Block Ordering Problem (BOP). The BOP was inspired by DNA sequencing technology that generates whole-genome sequencing data with relatively low coverage, inevitably resulting in the construction of a fragmented genome comprised of sequenced contigs whose relative ordering cannot be inferred directly. In particular, the problem is defined as follows: given two signed permutations (genomes) that are broken into blocks, order and orient each set of blocks in such a way that the number of cycles in the breakpoint graph of the resulting permutations is maximized, which they note "has been shown to approximate very well the [minimum] reversal distance between them."

In this work we use a more recently introduced measure of similarity between genomes as our objective criterion, namely the double-cut-and-join distance. Introduced in [59], the double-cut-and-join (DCJ) operation cuts a genome in two locations and then fuses the new ends in a different orientation. The authors present a linear-time algorithm for computing the DCJ distance between a pair of permutations using the breakpoint graph framework originally introduced in [5] to compute the reversal distance between a pair of permutations. [59] shows that the DCJ distance between genomes $\mathcal{A}$ and $\mathcal{B}$, $d_{DCJ}(\mathcal{A}, \mathcal{B})$, obeys:

$$d_{DCJ}(\mathcal{A}, \mathcal{B}) = b(\mathcal{A}, \mathcal{B}) - c(BG_{\mathcal{A},\mathcal{B}}), \tag{4.1}$$

where $b(\mathcal{A}, \mathcal{B})$ is the number of breakpoints and $c(BG_{\mathcal{A},\mathcal{B}})$ is the number of cycles in the

breakpoint graph of $\mathcal{A}$ and $\mathcal{B}$. They give a quadratic-time algorithm for computing a sorting sequence of DCJ operations based on the breakpoint graph of the two genomes. Interestingly, [59] observe that reversals, transpositions, and block interchanges with weights one, two, and two, respectively, can all be modeled by a single DCJ operation.

The DCJ operation was subsequently revisited by Bergeron et al. [9] who present a new framework for computing DCJ distance without using the breakpoint graph of [5]. Instead, in [9], the authors present a linear-time algorithm for computing the distance between two signed permutations with respect to the DCJ metric based on an *adjacency graph* construction. In particular, they show the following.

Theorem 7 (Bergeron et al., Thm 1)**.** *Given genomes $\mathcal{A}$ and $\mathcal{B}$ on the same set of N genes, the DCJ distance is:*

$$d_{DCJ}(\mathcal{A}, \mathcal{B}) = N - \left(c(AG_{\mathcal{A},\mathcal{B}}) + \frac{i(AG_{\mathcal{A},\mathcal{B}})}{2} \right), \tag{4.2}$$

where N is the number of genes, $c(AG_{\mathcal{A},\mathcal{B}})$ is the number of cycles in the adjacency graph and $i(AG_{\mathcal{A},\mathcal{B}})$ is the number of paths with an odd number of edges in the adjacency graph.

They also give a linear-time algorithm for computing a sorting sequence of DCJ operations.

The connection between reversals and DCJ operations when applied to a single circular chromosome was also described by [9]. In particular, a single DCJ operation when applied to a circular chromosome can result in a reversal or in a cycle fission (in which the circular chromosome becomes two circular chromosomes). Similarly, a DCJ operation on a pair of circular chromosomes can result in a cycle fusion (in which the 2 chromosomes become a single circular chromosome). Therefore, even if the start/end genomes are known to be circular-unichromosomal, a sequence of DCJ operations transforming one genome into another may create some intermediate genomes that are circular-multichromosomal.

Although the relationship between reversal operations and DCJ operations has been studied previously, here we make explicit the relationship between the breakpoint graph framework of [5] and the adjacency graph framework of [9]. In particular, we prove that the number of cycles in a breakpoint graph for a pair of permutations is equal to the number of cycles in the corresponding adjacency graph for the same pair. As a consequence, we can show that the algorithm given by [28] for the BOP yields a pair of genomes whose DCJ distance is minimum. The solution given in [28], however, is complicated; the algorithm relies on

a breakpoint graph framework and requires the construction first of a fragmented breakpoint graph (a generalization of the breakpoint graph requiring four colors to distinguish different types of vertices) and then of a block ordering graph (a vertex-bicolored and edge-bicolored graph constructed from the fragmented breakpoint graph). Finally, they give an algorithm to processes each type of component in the block ordering graph in succession. The algorithms we present below are simpler, relying on an adjacency graph framework.

Furthermore, the proof of correctness for the algorithm presented in [28] relies on some assumptions that are not proven explicitly. As stated above, the authors solve the BOP by ensuring the number of cycles in the resulting breakpoint graph is maximized. Given the input pair of partially assembled genomes, they construct a fragmented breakpoint graph (a generalization of a breakpoint graph) that exhibits, initially, some number of cycles. The argument then is that the total number of new cycles constructed by their algorithm is optimal. Although they quantify the number of new cycles that are constructed by their algorithm, they do not argue that this number is an upper bound and therefore optimal. (See Appendix C for additional discussion of the results presented in [28].) Here we prove the optimality of the algorithms we present.

4.2 Preliminaries

A *gene* is an oriented sequence of nucleotides, starting from its *tail* and ending at its *head*. The tail and head of a gene are referred to as its *extremities*. We denote the tail of a gene a as a_t and its head as a_h. For gene a, the extremities a_t and a_h are *obverse extremities*, denoted $a_t = \bar{a}_h$ and $a_h = \bar{a}_t$. When two genes appear consecutively on a chromosome, two of their extremities form an *adjacency*. An adjacency is represented as an unordered set of two extremities. For example, if genes a and b appear consecutively, they may form any of four possible adjacencies: $\{a_h, b_t\}, \{a_h, b_h\}, \{a_t, b_t\}, \{a_t, b_h\}$. An extremity that appears at the end of a linear chromosome and is therefore not adjacent to any other gene extremity is called a *telomere* and is represented as a singleton set, such as $\{a_t\}$.

A *genome* on N genes is a set of adjacencies and telomeres in which each of the $2N$ corresponding extremities is a member of exactly one element of the genome. The *genome graph* for genome $\mathcal{A}$, denoted $G_{\mathcal{A}}$, is a graph with nodes corresponding to the extremities in elements of $\mathcal{A}$ and edges between pairs of obverse extremities and extremities that are elements of the same adjacency in $\mathcal{A}$, i.e. the set of edges is $\{(u, v) \mid u = \bar{v}$ or $\{u, v\} \in \mathcal{A}\}$. Note that $G_{\mathcal{A}}$ is a graph that contains only nodes with degree one or two and that

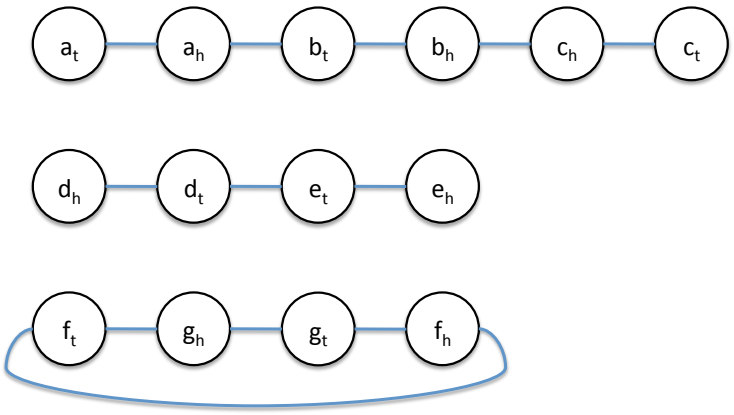

Figure 4.1: The genome graph for the genome $\mathcal{A}$. There are 14 nodes, representing the extremities of the 7 genes exhibited in $\mathcal{A}$. Each pair of obverse extremities defines an edge, and each adjacency in $\mathcal{A}$ defines an edge. The genome graph exhibits 3 connected components, corresponding to 3 constituent *chromosomes*. The top 2 chromosomes are *linear* and the bottom one is *circular*.

it may contain cycles. Given a genome $\mathcal{A}$ with N extremities, its genome graph can be constructed in $O(N)$ time in a straightforward manner.

A *chromosome* in a genome $\mathcal{A}$ is the subset of adjacencies and telomeres containing all the extremities in a single connected component in $G_{\mathcal{A}}$. The chromosomes of a genome can be inferred directly from its genome graph. A chromosome is *linear* if it contains exactly two telomeres and *circular* if it contains no telomeres. Note that a genome may contain either linear or circular chromosomes or a mixture of both.

For example, consider the following genome and its genome graph given in Fig. 4.1.

$$\mathcal{A} = \{\{a_t\}, \{a_h, b_t\}, \{b_h, c_h\}, \{c_t\}, \{d_h\}, \{d_t, e_t\}, \{e_h\}, \{f_t, g_h\}, \{g_t, f_h\}\}.$$

$\mathcal{A}$ contains seven genes and three chromosomes, two of which are linear and one of which is circular.

We also define a *partial genome* on N genes to be a set of adjacencies and telomeres in which each of the $2N$ extremities can be a member of no more than one element. Similarly, the *partial genome graph* for a partial genome $\mathcal{X}$ on N nodes is the graph induced on the $2N$ corresponding extremities of the N genes with edges between pairs of obverse extremities and pairs of extremities that are elements of an adjacency in $\mathcal{X}$. See Figure 4.2 for an example.

Definition 23. *A genome is* ***circular-unichromosomal*** *if and only if it contains exactly one chromosome and no telomeres.*

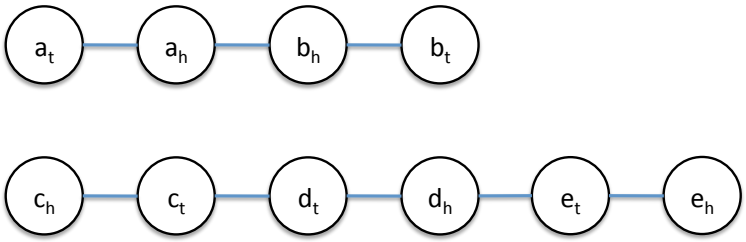

Figure 4.2: A partial genome graph for the partial genome $\mathcal{X} = \{\{a_h, b_h\}, \{c_t, d_t\}, \{d_h, e_t\}\}$ on genes a, b, c, d, e. The excluded adjacencies are $\mathcal{E}(\mathcal{X}) = \{\{a_t, b_t\}, \{c_h, e_h\}\}$.

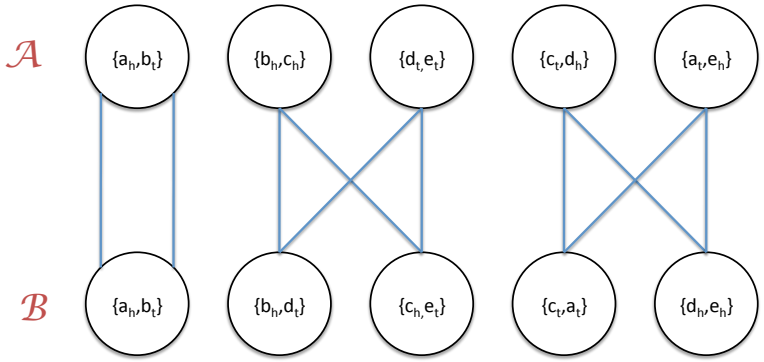

Figure 4.3: The adjacency graph for genomes $\mathcal{A} = \{\{a_h, b_t\}, \{b_h, c_h\}, \{d_t, e_t\}, \{c_t, d_h\}, \{a_t, e_h\}\}$ and $\mathcal{B} = \{\{a_h, b_t\}, \{b_h, d_t\}, \{c_h, e_t\}, \{c_t, a_t\}, \{d_h, e_h\}\}$. The graph contains 3 cycles.

For a pair of genomes, $\mathcal{A}$ and $\mathcal{B}$, that contain the same set of genes, we define the *adjacency graph*, denoted $AG_{\mathcal{AB}}$, as the bipartite graph whose vertices correspond to the elements of $\mathcal{A}$ and $\mathcal{B}$. For the pair $a \in \mathcal{A}$, $b \in \mathcal{B}$, there are $\mid a \cap b \mid$ edges between a and b in the adjacency graph. Therefore, a node in an adjacency graph may have degree one (for a telomere) or two (for an adjacency). An adjacency graph can contain only simple paths and simple cycles. Note that an adjacency graph may contain parallel edges (or a simple cycle of length two). The adjacency graph for a pair of genomes on N genes can contain at most N cycles. See Fig. 4.3. Given a pair of genomes on N genes, we can construct their adjacency graph in $O(N)$ time in a straightforward manner.

Similarly, we can define a *partial adjacency graph* for a pair of partial genomes or one complete genome and one partial genome. See Figure 4.4 for an example. Note that a node may have degree 0 in a partial adjacency graph.

In [9], the authors proved that the DCJ distance between a pair of genomes defined on the same set of N genes can be computed by analyzing the adjacency graph. (See Thm 7.)

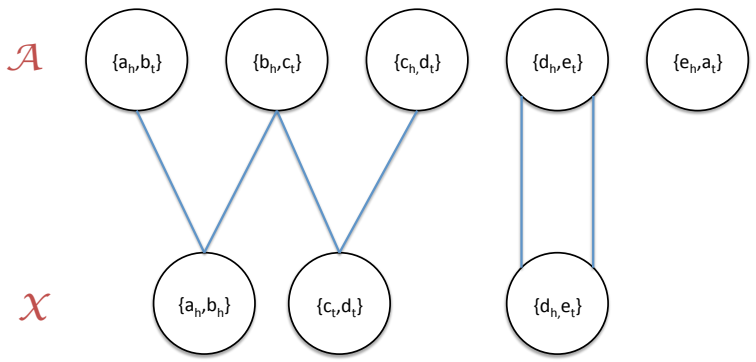

Figure 4.4: A partial adjacency graph for the genome $\mathcal{A} = \{\{a_h, b_t\}, \{b_h, c_t\}, \{c_h, d_t\}, \{d_h, e_t\}, \{e_h, a_t\}\}$ and partial genome $\mathcal{X} = \{\{a_h, b_h\}, \{c_t, d_t\}, \{d_h, e_t\}\}$ on genes a, b, c, d, e. The missing extremities are $\mathcal{M}(\mathcal{X}) = \{a_t, b_t, c_h, e_h\}$. The unsatisfied pairs are $\mathcal{U}(\mathcal{A}, \mathcal{X}) = \{\{b_t, c_h\}, \{e_h, a_t\}\}$.

4.3 The Breakpoint Graph and the Adjacency Graph

In the previous sections, we described the adjacency graph for a pair of genomes. Here we present the *breakpoint graph* structure introduced by [5]. We only discuss the breakpoint graph structure for pairs of circular-unichromosomal genomes, although the structure can be generalized for linear-unichromosomal genomes as well. First, in [5], the authors assume that genomes are represented by signed permutations on an alphabet of genes. A signed permutation π can be transformed into the equivalent set representation of a genome $\mathcal{G}_\pi$ described in Section 4.2. First, we transform π into an unsigned permutation π' on $2N$ markers: for each gene in π with positive orientation, such as $+a$, replace it in π' by the consecutive pair of extremities a_t, a_h, for each gene in π with negative orientation, such as $-a$, replace it in π' by the consecutive pair of extremities a_h, a_t. Then, for each pair of successive extremities i, j such that i and j are not obverse extremities (i.e. $i \neq \bar{j}$) in π', add the adjacency $\{i, j\}$ to $\mathcal{G}_\pi$. Also add an adjacency corresponding to the first and last extremities in π'.

A breakpoint graph for a pair $\mathcal{A}, \mathcal{B}$ of circular-unichromosomal genomes on N genes contains $2N$ nodes – one for each extremity. The breakpoint graph contains three types of edges. Pairs of obverse extremities are connected with *obverse* edges. For every adjacency $\{i, j\} \in \mathcal{A}$, i and j are connected with a *black* edge. For every adjacency $\{i', j'\} \in \mathcal{B}$, i' and j' are connected with a *green* edge. A *cycle* in the breakpoint graph is a green-black alternating cycle. See Figure 4.5 for an example.

In this section, we show explicitly the connection between the breakpoint graph and the

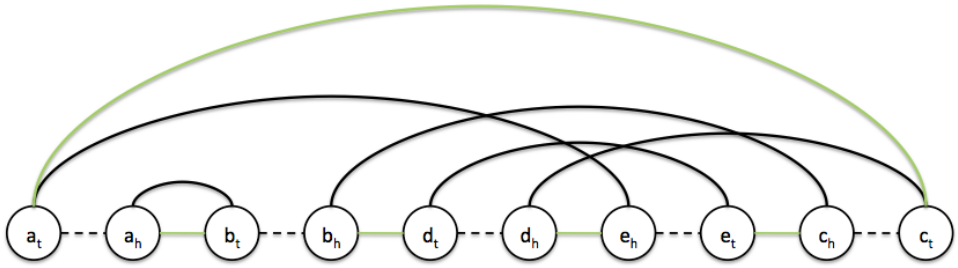

Figure 4.5: The breakpoint graph for genomes $\mathcal{A} = \{\{a_h, b_t\}, \{b_h, c_h\}, \{d_t, e_t\}, \{c_t, d_h\}, \{a_t, e_h\}\}$ and $\mathcal{B} = \{\{a_h, b_t\}, \{b_h, d_t\}, \{c_h, e_t\}, \{c_t, a_t\}, \{d_h, e_h\}\}$. Black edges correspond to adjacencies in $\mathcal{A}$, and green edges correspond to adjacencies in $\mathcal{B}$. Dashed edges represent obverse edges. There are three green-black alternating cycles.

adjacency graph for a pair of genomes: we prove that the number of cycles in the breakpoint graph for a pair of unichromosomal genomes is equivalent to the number of cycles in their adjacency graph. That is, we prove that for unichromosomal genomes $\mathcal{A}$ and $\mathcal{B}$, $c(BP_{\mathcal{A},\mathcal{B}}) = c(AG_{\mathcal{A},\mathcal{B}})$. To our knowledge, this connection between the two graphs has not been shown explicitly before.

Lemma 7. *Given a pair of circular-unichromosomal genomes, $\mathcal{A}$, $\mathcal{B}$, on N genes, the number of cycles in the breakpoint graph equals the number of cycles in the adjacency graph.*

Proof: To prove the equivalence of the two graphs, we give a series of cycle-preserving graph transformation operations that converts the adjacency graph into the breakpoint graph. Start with the adjacency graph $AG_{\mathcal{A},\mathcal{B}}$. For each adjacency in $\mathcal{A}$, split the node into two – corresponding to the two extremities – and connect them with a black edge. For each adjacency in $\mathcal{B}$, split the node into two – corresponding to the two extremities – and connect them with a green edge. In both $\mathcal{A}$ and $\mathcal{B}$, connect obverse extremities with obverse edges. (Note that the set of vertices in $\mathcal{A}$ is equal to the set of vertices in $\mathcal{B}$, and the obverse edges between extremities in $\mathcal{A}$ will be the same as the set of obverse edges between extremities in $\mathcal{B}$.) We then merge all identical pairs of vertices from $\mathcal{A}$ and $\mathcal{B}$. We delete one of the two parallel copies of each obverse edge. The resulting graph is a breakpoint graph for $\mathcal{A}$ and $\mathcal{B}$ and there is a one-to-one correspondence between cycles in the original adjacency graph and cycles in the resulting breakpoint graph. ■

In [30], the authors show that for circular-unichromosomal genomes $\mathcal{A}$ and $\mathcal{B}$ on N genes, the reversal distance obeys:

$$d_{reversal}(\mathcal{A}, \mathcal{B}) = N - c(BP_{\mathcal{A},\mathcal{B}}) + t, \tag{4.3}$$

where $c(BP_{\mathcal{A},\mathcal{B}})$ is the number of cycles in the breakpoint graph and t is usually a small constant that accounts for the number of operations needed to handle the so-called hurdles and fortresses in the breakpoint graph. By comparison, [9] show that the DCJ distance obeys:

$$d_{DCJ}(\mathcal{A},\mathcal{B}) = N - c(AG_{\mathcal{A},\mathcal{B}}), \tag{4.4}$$

where $c(AG_{\mathcal{A},\mathcal{B}})$ is the number of cycles in the adjacency graph. Therefore, Lemma 7 implies that the only difference between the reversal distance and the DCJ distance for a pair of genomes owes to the existence of hurdles or fortresses in the breakpoint graph and can be quantified by t.

Another consequence of Lemma 7 is that the goal of solving the BOP with respect to the number of cycles in the breakpoint graph is equivalent to solving the BOP with respect to the number of cycles in the adjacency graph. Hence, although they state that their goal is to minimize the approximate reversal distance between the two final genomes, the algorithm of [28] maximizes the number of cycles in their adjacency graph and thus also minimizes the DCJ distance between them.

4.4 Problem Formulation

Here, we use the formulation of [9] and an adjacency-graph approach to simplify the result of [28] to solve the block ordering problem with respect to the DCJ metric. In our discussion, we consider a special case of the problem, that we call the *Completion Problem*, in which one of the two input genomes is a fully sequenced reference genome, and the goal is to complete a partially assembled genome so as to maximize its similarity to the known reference genome. We describe two variants of the completion problem – one in which the form of the final assembled genome is unconstrained and one in which the final genome is required to be comprised of a single, circular chromosome. We note that our algorithms for the completion problem can be extended to solve the BOP more generally.

We begin with some definitions.

Definition 24. *A **completion** of a partial genome $\mathcal{X}$ is a genome $\hat{\mathcal{X}}$ such that $\mathcal{X} \subseteq \hat{\mathcal{X}}$.*

Definition 25. *A **circular-unichromosomal completion** of a partial genome $\mathcal{X}$ is a genome $\hat{\mathcal{X}}$ such that $\mathcal{X} \subseteq \hat{\mathcal{X}}$ and $\hat{\mathcal{X}}$ is circular-unichromosomal.*

Definition 26. *Given a circular-unichromosomal genome $\mathcal{A}$ on N genes and a partial genome $\mathcal{X}$ on N genes containing only adjacencies, the **Unrestricted Completion Problem** is to find a completion $\hat{\mathcal{X}}$ of $\mathcal{X}$ such that $d_{DCJ}(\mathcal{A},\hat{\mathcal{X}})$ is minimum.*

Definition 27. *Given a circular-unichromosomal genome $\mathcal{A}$ on N genes and a partial genome $\mathcal{X}$ on N genes containing only adjacencies, the* ***Restricted Completion Problem*** *is to find a circular-unichromosomal completion $\hat{\mathcal{X}}$ of $\mathcal{X}$ such that $d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}})$ is minimum.*

We note that, as its name suggests, the restricted version of the problem is more constrained than the unrestricted version.

Claim 1. *Given a circular-unichromosomal genome $\mathcal{A}$, an optimal solution $\hat{\mathcal{X}}_U$ to the unrestricted completion problem, and an optimal solution $\hat{\mathcal{X}}_R$ to the restricted completion problem, $d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}}_U) \leq d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}}_R)$.*

Proof: Any circular-unichromosomal completion $\hat{\mathcal{X}}$ of $\mathcal{X}$ is also a completion of $\mathcal{X}$. Therefore, an optimal solution $\hat{\mathcal{X}}_R$ to the restricted version of the completion problem is also a (possibly not optimal) solution to the unrestricted version of the problem. Thus, the distance $d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}}_R)$ is an upper bound on the distance $d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}}_U)$ to an optimal solution to the unrestricted version of the problem.

4.5 The Algorithm

4.5.1 The Unrestricted Problem

First, we give an algorithm for the unrestricted version of the completion problem. Our input to the problem is a reference genome $\mathcal{A}$ that is circular-unichromosomal and a set of measured adjacencies $\mathcal{X}$ that contain each extremity that appears in an element of $\mathcal{A}$ at most once. The goal is to complete the partial genome $\mathcal{X}$ by adding new adjacencies and telomeres to it in order to build the full genome that is most similar to $\mathcal{A}$ with respect to DCJ distance.

By Thm. 7, any strategy that will complete a partial genome $\mathcal{X}$ in such a way that the DCJ distance between the completion of that genome $\hat{\mathcal{X}}$ and a given reference genome $\mathcal{A}$ will be minimal will maximize the value of $c(AG_{\mathcal{A},\hat{\mathcal{X}}}) + i(AG_{\mathcal{A},\hat{\mathcal{X}}})/2$ in the resulting adjacency graph.

Let us consider the partial adjacency graph $AG_{\mathcal{A},\mathcal{X}}$ for $\mathcal{A}$ and $\mathcal{X}$. Recall that it is bipartite. Note that because $\mathcal{A}$ is circular-unichromosomal, it contains only adjacencies – no telomeres. Also note that $\mathcal{X}$ is defined to be a set of adjacencies. Moreover, because $\mathcal{A}$ is a complete genome, all the $2N$ extremities are represented, so all the nodes in $\mathcal{X}$ will have degree 2 in the partial adjacency graph. In other words, the partial adjacency graph must contain exactly $2 \mid \mathcal{X} \mid$ edges. Consequently, the partial adjacency graph contains only

cycles of even length and even-length paths whose two endpoints are in $\mathcal{A}$.

There are $\mid \mathcal{X} \mid$ adjacencies in $\mathcal{X}$, and because each adjacency contains two extremities, that means that $2 \mid \mathcal{X} \mid$ of the $2N$ total extremities are represented in $\mathcal{X}$. The rest are missing.

Definition 28. *Let $\mathcal{M}(\mathcal{X})$ be the set of $2N - 2 \mid \mathcal{X} \mid$* ***missing extremities*** *that do not appear in $\mathcal{X}$.*

For example, in Figure 4.4, $\mathcal{M}(\mathcal{X}) = \{a_t, b_t, c_h, e_h\}$. These are the extremities that must be added to $\mathcal{X}$ in order to complete it.

Claim 2. *Let $\mathcal{X}$ be a partial genome on N genes, and let $\mathcal{P}$ be any partition of the elements of $\mathcal{M}(\mathcal{X})$ into $\frac{|\mathcal{M}(\mathcal{X})|}{2} = N - \mid \mathcal{X} \mid$ adjacencies. $\mathcal{X} \cup \mathcal{P}$ is a completion of $\mathcal{X}$.*

The claim follows from the definition of a completion; the set $\mathcal{X} \cup \mathcal{P}$ contains $\mathcal{X}$ and exhibits all $2N$ extremities.

Claim 3. *Given a partial genome $\mathcal{X}$ and a perfect matching M on $\mathcal{M}(X)$, $\mathcal{X} \cup M$ is a completion of $\mathcal{X}$.*

The claim follows directly from Claim 2; the perfect matching M is a partition of $\mathcal{M}(\mathcal{X})$ into $\frac{|\mathcal{M}(\mathcal{X})|}{2}$ adjacencies.

Consider again the partial adjacency graph. Note that an extremity $m \in \mathcal{M}(\mathcal{X})$ must appear as a member of a node $a \in \mathcal{A}$ that has degree zero or one in $AG_{\mathcal{A},\mathcal{X}}$.

Definition 29. *Given a circular-unichromosomal genome $\mathcal{A}$, and a partial genome $\mathcal{X}$, let u, v be extremities in $\mathcal{M}(\mathcal{X})$. $\{u, v\}$ is an* ***unsatisfied pair*** *provided either:*

1. *$\{u, v\} \in \mathcal{A}$, or*
2. *there exist u', v' such that $\{u, u'\}$ and $\{v, v'\}$ are degree-one nodes at opposite ends of an even path in $AG_{\mathcal{A},\mathcal{X}}$.*

See Fig. 4.4 for an example.

Definition 30. *Given a circular-unichromosomal genome $\mathcal{A}$, and a partial genome $\mathcal{X}$, $\mathcal{U}(\mathcal{A}, \mathcal{X})$ is the set of all unsatisfied pairs of extremities.*

Note that $\mid \mathcal{U}(\mathcal{A}, \mathcal{X}) \mid = N - \mid \mathcal{X} \mid = \frac{|\mathcal{M}(\mathcal{X})|}{2}$; every extremity in $\mathcal{M}(\mathcal{X})$ is a member of an unsatisfied pair.

Claim 4. *Given a circular-unichromosomal genome $\mathcal{A}$, a partial genome $\mathcal{X}$, and an unsatisfied pair $\{u, v\} \in \mathcal{U}(\mathcal{A}, \mathcal{X})$, the partial genome $\mathcal{X}' = \mathcal{X} \cup \{\{u, v\}\}$ will exhibit a partial adjacency graph such that $c(AG_{\mathcal{A},\mathcal{X}'}) = c(AG_{\mathcal{A},\mathcal{X}}) + 1$, where $c(AG_{\mathcal{A},\mathcal{X}})$ is the number of cycles in the partial adjacency graph for $\mathcal{A}$ and $\mathcal{X}$ (and similar for $c(AG_{\mathcal{A},\mathcal{X}'})$).*

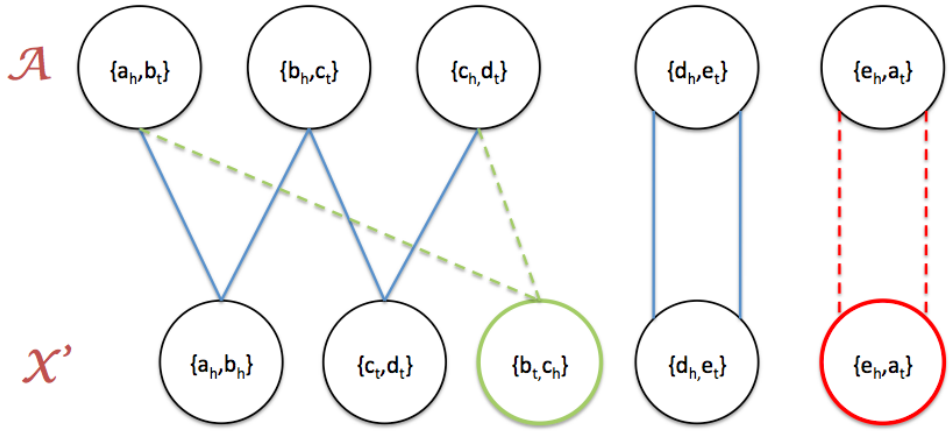

Figure 4.6: An illustration of Claim 4. The partial adjacency graph from Fig. 4.4 is augmented to show both of the possible cases for processing an unsatisfied pair. The red node and edges illustrate the first case – adding an adjacency to $\mathcal{X}$ that corresponds to an unsatisfied pair that is also an adjacency in $\mathcal{A}$, yielding a new cycle of length two. The green node and edges illustrate the second case – adding an adjacency to $\mathcal{X}$ that corresponds to an unsatisfied pair that appear at opposite ends of an even path, transforming the even path into a new cycle.

Proof: Let $AG_{\mathcal{A},\mathcal{X}}$ be the partial adjacency graph for $\mathcal{A}$ and $\mathcal{X}$. Because $\{u, v\}$ is an unsatisfied pair, there are two possible cases: either (1) $\{u, v\} \in \mathcal{A}$ or (2) there exist u', v' such that $\{u, u'\}$ and $\{v, v'\}$ are degree-one nodes at opposite ends of a path with l edges in $AG_{\mathcal{A},\mathcal{X}}$ where l is even. (In the partial adjacency graph in Fig. 4.4, the unsatisfied pair $\{e_h, a_t\}$ is an example corresponding to the first case, and the unsatisfied pair $\{b_t, c_h\}$ is an unsatisfied pair corresponding to the second case.) Suppose we add $\{u, v\}$ as a new adjacency to $\mathcal{X}$, yielding $\mathcal{X}'$. In the first case, the node $\{u, v\}$ in $\mathcal{A}$ will have degree zero in $AG_{\mathcal{A},\mathcal{X}}$. Therefore, adding $\{u, v\}$ to $\mathcal{X}$ will create a new cycle of length two in the resulting adjacency graph $AG_{\mathcal{A},\mathcal{X}'}$. In the second case, adding $\{u, v\}$ to $\mathcal{X}$ as a new adjacency, will induce edges between $\{u, v\}$ in $\mathcal{X}'$ and $\{u, u'\}$ in $\mathcal{A}$ and between $\{u, v\}$ in $\mathcal{X}'$ and $\{v, v'\}$ in $\mathcal{A}$, transforming the length-l path between $\{u, u'\}$ and $\{v, v'\}$ into a cycle with $l + 2$ edges in $AG_{\mathcal{A},\mathcal{X}'}$. (See Fig. 4.6 for an illustration.) ■

As noted above, for the unrestricted completion problem, an optimal completion $\hat{\mathcal{X}}$ of partial genome $\mathcal{X}$ must exhibit a maximum number of cycles and odd paths in the resulting adjacency graph for the circular-unichromosomal reference genome $\mathcal{A}$ and the completion $\hat{\mathcal{X}}$, $AG_{\mathcal{A},\hat{\mathcal{X}}}$. Recall that a partial adjacency graph for a circular-unichromosomal genome and a partial genome containing only adjacencies can contain only cycles and paths with an even number of edges. Also recall that we may complete $\mathcal{X}$ by choosing a partition of the elements of $\mathcal{M}(\mathcal{X})$ into a perfect matching of size $N - \mid \mathcal{X} \mid$ and then adding those

elements of the matching as new adjacencies in $\hat{\mathcal{X}}$ (by Claim 3). Note that the $N- \mid \mathcal{X} \mid$ elements of $\mathcal{U}(\mathcal{A}, \mathcal{X})$ are a perfect matching on $\mathcal{M}(\mathcal{X})$. Thus, $\hat{\mathcal{X}} = \mathcal{X} \cup \mathcal{U}(\mathcal{A}, \mathcal{X})$ is a completion of $\mathcal{X}$. Furthermore, by Claim 4, $\hat{\mathcal{X}}$ contains $c(AG_{\mathcal{A},\mathcal{X}})+ \mid \mathcal{U}(\mathcal{A}, \mathcal{X}) \mid$ cycles, where $c(AG_{\mathcal{A},\mathcal{X}})$ denotes the number of cycles in the partial adjacency graph for $\mathcal{A}$ and $\mathcal{X}$. This is maximum because of the following bound.

Claim 5. *Let $\mathcal{A}$ be a circular-unichromosomal genome and $\mathcal{A}$ be a partial genome. For any completion $\hat{\mathcal{X}}$, $c(AG_{\mathcal{A},\hat{\mathcal{X}}}) - c(AG_{\mathcal{A},\mathcal{X}}) \leq \frac{|\mathcal{M}(\mathcal{X})|}{2} =\mid \mathcal{U}(\mathcal{A}, \mathcal{X}) \mid$.*

The claim follows from the fact that adding a new adjacency to $\mathcal{X}$ requires the selection of two extremities from $\mathcal{M}(\mathcal{X})$ and each new adjacency can increase the number of cycles in the adjacency graph by at most one.

Thus, there is a straightforward algorithm for solving the unrestricted version of the completion problem: construct the partial adjacency graph for the reference genome $\mathcal{A}$ and the partial genome $\mathcal{X}$, identify the unsatisfied pairs $\mathcal{U}(\mathcal{A}, \mathcal{X})$, and add the unsatisfied pairs to $\mathcal{X}$ to complete the genome $\hat{\mathcal{X}} = \mathcal{X} \cup \mathcal{U}(\mathcal{A}, \mathcal{X})$. The running time is linear in the number of genes, N. Every unsatisfied pair will yield a new cycle in the resulting adjacency graph $AG_{\mathcal{A},\hat{\mathcal{X}}}$, and this is optimal by Claim 5.

Theorem 8 (Unrestricted Completion Problem)**.** *Given a circular-unichromosomal genome $\mathcal{A}$ and partial genome $\mathcal{X}$ on N genes, an optimal solution to the unrestricted completion problem $\hat{\mathcal{X}}$ of $\mathcal{X}$ will exhibit the DCJ distance:*

$$\begin{aligned} d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}}) &= N - [c(AG_{\mathcal{A},\mathcal{X}})+ \mid \mathcal{U}(\mathcal{A}, \mathcal{X} \mid], \\ &= N - [c(AG_{\mathcal{A},\mathcal{X}}) + (N- \mid \mathcal{X} \mid)], \\ &=\mid \mathcal{X} \mid -c(AG_{\mathcal{A},\mathcal{X}}). \end{aligned} \tag{4.5}$$

where $c(AG_{\mathcal{A},\mathcal{X}})$ is the number of cycles in the partial adjacency graph $AG(\mathcal{A}, \mathcal{X})$.

We note that a solution to the unrestricted version of the problem can exhibit as many as $\Omega(N)$ chromosomes. Allowing the construction of a genome to include arbitrarily many chromosomes may not be desirable from a biological perspective. Instead, we may choose to enforce that the constructed genome contain, for example, a single chromosome. In the next section, we consider this more restricted version of the completion problem.

4.5.2 The Restricted Problem

Here we give an algorithm for solving the restricted completion problem. The input to this problem is the same as for the unrestricted version. We can assume again that the input

reference genome $\mathcal{A}$ only contains adjacencies because it is circular-unichromosomal and that the input partial genome $\mathcal{X}$ only contains adjacencies by definition. Therefore, as before, the partial adjacency graph $AG_{\mathcal{A},\mathcal{X}}$ is comprised of only cycles of even length and even-length paths that start and end at nodes in $\mathcal{A}$.

In the restricted completion problem, we must complete $\mathcal{X}$ such that the resulting genome $\hat{\mathcal{X}}$ is circular-unichromosomal. The definition of a circular-unichromosomal genome is that it contains no telomeres and only one chromosome. The genome graph of a genome that contains only one chromosome contains only a single connected component. Therefore, we have the following.

Claim 6. *The genome graph of a circular-unichromosomal genome on N genes is comprised of a single simple cycle that visits all the nodes (corresponding to all of the $2N$ extremities).*

Consider the partial genome graph $G_{\mathcal{X}}$. Any completion $\hat{\mathcal{X}}$ of $\mathcal{X}$ will contain $G_{\mathcal{X}}$ as a subgraph because all the original adjacencies in $\mathcal{X}$ will remain in the final completion $\hat{\mathcal{X}}$. If $\mathcal{X}$ is a valid input (i.e. it can be completed in such a way that the resulting genome is circular-unichromosomal), we can assume that $G_{\mathcal{X}}$ does not contain any cycles. Instead, $G_{\mathcal{X}}$ is comprised of a collection of simple paths.

Definition 31. *Given a partial genome $\mathcal{X}$ with missing extremities $\mathcal{M}(\mathcal{X})$, let u, v be extremities in $\mathcal{M}(\mathcal{X})$. $\{u, v\}$ is an **excluded adjacency** if u and v are both degree-one nodes at opposite ends of the same simple path in the partial genome graph $G_{\mathcal{X}}$.*

Definition 32. *Given a partial genome $\mathcal{X}$, $\mathcal{E}(\mathcal{X})$ is the set of all excluded adjacencies.*

See Fig. 4.2 for an example.

Lemma 8. *For a partial genome $\mathcal{X}$ comprised of only adjacencies with genome graph $G_{\mathcal{X}} = (V, E)$ and a perfect matching M on $\mathcal{M}(\mathcal{X})$, the completion $\hat{\mathcal{X}} = \mathcal{X} \cup M$ is circular-unichromosomal if and only if $G_{\hat{\mathcal{X}}} = (V, E \cup M)$ is a simple cycle.*

Proof: The forward direction of the lemma is a consequence of Claim 6. For the backward direction, suppose $G_{\hat{\mathcal{X}}}$ is a simple cycle. Then every extremity in $G_{\hat{\mathcal{X}}}$ has degree two and belongs to the same the connected component. Therefore, $G_{\hat{\mathcal{X}}}$ has exactly one chromosome and no telomeres and is, thus, circular-unichromosomal. ■

Lemma 9. *Given a partial genome $\mathcal{X}$ that contains only adjacencies, let M be a perfect matching on $\mathcal{M}(\mathcal{X})$. For a partition of $\mathcal{E}(\mathcal{X})$ into two sets $\mathcal{E}^1(\mathcal{X})$ and $\mathcal{E}^2(\mathcal{X})$, let $\mathcal{M}^1(\mathcal{X})$ denote the subset of $\mathcal{M}(\mathcal{X})$ equal to $\mathcal{M}^1(\mathcal{X}) = \{u, v \in \mathcal{M}(\mathcal{X}) \mid \{u, v\} \in \mathcal{E}^1(\mathcal{X})\}$, and similar for $\mathcal{M}^2(\mathcal{X})$. There exists a partition of $\mathcal{E}(\mathcal{X})$ into two nonempty sets $\mathcal{E}^1(\mathcal{X})$*

and $\mathcal{E}^2(\mathcal{X})$ such that M can be partitioned into a perfect matching M^1 on $\mathcal{M}^1(\mathcal{X})$ and a perfect matching M^2 on $\mathcal{M}^2(\mathcal{X})$, if and only if the completion $\hat{\mathcal{X}} = \mathcal{X} \cup M$ is not circular-unichromosomal.

Proof: First we prove the forward direction. Suppose M can be partitioned into M^1, a perfect matching on $\mathcal{M}^1(\mathcal{X})$, and M^2, a perfect matching on $\mathcal{M}^2(\mathcal{X})$. By the definition of an excluded adjacency, for every $\{u, v\} \in \mathcal{E}(\mathcal{X})$, the partial genome graph $G_{\mathcal{X}} = (V, E)$ contains a u-to-v path. Therefore, the partial genome graph induced on $(V, E \cup M^1)$ contains a cycle that visits every extremity in $\mathcal{M}^1(\mathcal{X})$. And similarly, the partial genome graph induced on $(V, E \cup M^2)$ contains a cycle that visits every extremity in $\mathcal{M}^2(\mathcal{X})$. Thus, the genome graph $G_{\hat{\mathcal{X}}} = (V, E \cup M^1 \cup M^2)$ contains two edge-disjoint cycles. Therefore, by Lemma 8, $\hat{\mathcal{X}}$ is not circular-unichromosomal.

For the backward direction, suppose the completion $\hat{\mathcal{X}} = \mathcal{X} \cup M$ is not circular-unichromosomal. We shall show the existence of a partition of $\mathcal{E}(\mathcal{X})$ into two sets for which M can be partitioned into two perfect matchings. By Lemma 8, the genome graph $G_{\hat{\mathcal{X}}}$ is not a simple cycle. But because $\mathcal{X}$ contains only adjacencies and M is a perfect matching, every node in $G_{\hat{\mathcal{X}}}$ has degree two and, thus, it must be comprised of a collection of simple cycles. Consider one such cycle C_1. Let $\mathcal{M}^1(\mathcal{X})$ be the subset of $\mathcal{M}(\mathcal{X})$ visited by C_1. Note that for every $\{u, v\} \in \mathcal{E}(\mathcal{X})$, if u is in $\mathcal{M}^1(\mathcal{X})$, then v is in $\mathcal{M}^1(\mathcal{X})$ and C_1 contains a u-to-v subpath that does not contain any other element of $\mathcal{M}^1(\mathcal{X})$. Let $\mathcal{E}^1(\mathcal{X})$ be the subset $\mathcal{E}^1(\mathcal{X}) = \{\{u, v\} \in \mathcal{E}(\mathcal{X}) \mid \{u, v\} \in \mathcal{M}^1(\mathcal{X})\}$. Also let M^1 be the matching on $\mathcal{M}^1(\mathcal{X})$ defined by the set of edges in C_1 that do not appear in $G_{\mathcal{X}}$. Let $\mathcal{M}^2(\mathcal{X})$ be the subset of $\mathcal{M}(\mathcal{X})$ visited by every cycle in $G_{\hat{\mathcal{X}}}$ other than C_1, i.e. $\mathcal{M}^2(\mathcal{X}) = \mathcal{M}(\mathcal{X}) \setminus \mathcal{M}^1(\mathcal{X})$. Let M^2 be the matching on $\mathcal{M}^2(\mathcal{X})$ defined by the set of edges in those other cycles that do not appear in $G_{\mathcal{X}}$. M^1 and M^2 partition M, and $\mathcal{E}^1(\mathcal{X})$ and $\mathcal{E}^2(\mathcal{X})$ partition $\mathcal{E}(\mathcal{X})$. Moreover, M^1 is a perfect matching on $\mathcal{M}^1(\mathcal{X})$ and M^2 is a perfect matching on $\mathcal{M}^2(\mathcal{X})$. ■

We state the contrapositive of the backward direction of Lemma 9 as a corollary.

Corollary 9. *Given a partial genome $\mathcal{X}$ that contains only adjacencies, let M be a perfect matching on $\mathcal{M}(\mathcal{X})$. If there does not exist a partition of $\mathcal{E}(\mathcal{X})$ into two sets $\mathcal{E}^1(\mathcal{X})$ and $\mathcal{E}^2(\mathcal{X})$, such that M can be partitioned into perfect matchings on $\mathcal{E}^1(\mathcal{X})$ and $\mathcal{E}^2(\mathcal{X})$, then the completion $\hat{\mathcal{X}} = \mathcal{X} \cup M$ is circular-unichromosomal.*

This corollary characterizes the types of adjacencies we can use to augment a partial genome $\mathcal{X}$ in order to construct a circular-unichromosomal completion. In particular, we must find a matching M on $\mathcal{M}(\mathcal{X})$ and completion $\hat{\mathcal{X}} = \mathcal{X} \cup M$ such that for each

$\{u, v\} \in \mathcal{E}(\mathcal{X})$, $G_{\hat{\mathcal{X}}}$ contains a simple u-to-v path that contains every edge in M.

Consider the augmentation of $\mathcal{X}$ by a single new adjacency $\{u, v\}$ on elements of $\mathcal{M}(\mathcal{X})$, yielding the new partial genome $\mathcal{X}'$. If the pair $\{u, v\}$ is not an excluded adjacency, then the addition of the new adjacency merges two contigs into one longer contig. Suppose the pairs $\{u, u'\}$ and $\{v, v'\}$ were excluded adjacencies in $\mathcal{E}(\mathcal{X})$. The partial genome $\mathcal{X}'$ will contain one fewer contig and the set of excluded adjacencies will now contain the extremities found at opposite ends of that merged contig, namely $\{u', v'\}$. u and v will no longer belong to any excluded adjacencies because they are not elements of $\mathcal{M}(\mathcal{X}')$.

Recall that the objective of the restricted block ordering problem is to find a completion $\hat{\mathcal{X}}$ of $\mathcal{X}$ that maximizes the number of cycles in the resulting adjacency graph for the reference genome $\mathcal{A}$ and $\hat{\mathcal{X}}$. Recall too that augmenting $\mathcal{X}$ with a perfect matching on $\mathcal{M}(\mathcal{X})$ induces a number of new cycles in the adjacency graph without changing the number of cycles that exist in the partial adjacency graph $AG_{\mathcal{A},\mathcal{X}}$. Again, the number of such new cycles is bounded by $\mid \mathcal{U}(\mathcal{A}, \mathcal{X}) \mid = \mid \mathcal{M}(\mathcal{X}) \mid /2$. In the unrestricted version of the problem, we were able to achieve this upper bound by adding the adjacencies defined by $\mathcal{U}(\mathcal{A}, \mathcal{X})$ to $\mathcal{X}$. However, Lemma 9 limits our ability to select arbitrarily a set of adjacencies from $\mathcal{M}(\mathcal{X})$ to add to $\mathcal{X}$. In particular, if the genome graph $G_{\mathcal{X} \cup \mathcal{U}(\mathcal{A},\mathcal{X})}$ contains more than one cycle, the completion $\mathcal{X} \cup \mathcal{U}(\mathcal{A}, \mathcal{X})$ is not circular-unichromosomal. For every cycle in the genome graph for such a multi-chromosomal completion, at least one edge cannot appear in a circular-unichromosomal completion.

Lemma 10. *Given a circular-unichromosomal genome $\mathcal{A}$ and a set of adjacencies $\mathcal{X}$, let $\hat{\mathcal{X}}$ be a circular-unichromosomal completion of $\mathcal{X}$. The number of cycles in the adjacency graph for $\mathcal{A}$ and $\hat{\mathcal{X}}$ is bounded by:*

$$c(AG_{\mathcal{A},\hat{\mathcal{X}}}) \leq c(AG_{\mathcal{A},\mathcal{X}}) + \frac{\mid \mathcal{M}(\mathcal{X}) \mid}{2} - c(G_{\mathcal{X} \cup \mathcal{U}(\mathcal{A},\mathcal{X})}) + 1. \tag{4.6}$$

Proof: First, suppose $c(G_{\mathcal{X} \cup \mathcal{U}(\mathcal{A},\mathcal{X})}) = 1$. In this case, the lemma follows from the fact that the maximum number of cycles that can be added to $AG_{\mathcal{A},\mathcal{X}}$ by completing $\mathcal{X}$ is bounded by $\frac{|\mathcal{M}(\mathcal{X})|}{2} = \mid \mathcal{U}(\mathcal{A}, \mathcal{X}) \mid$. Now suppose $c(G_{\mathcal{X} \cup \mathcal{U}(\mathcal{A},\mathcal{X})}) > 1$. Then the completion $\mathcal{X}' = \mathcal{X} \cup \mathcal{U}(\mathcal{A}, \mathcal{X})$ is not circular-unichromosomal. So, $\hat{\mathcal{X}}$ cannot contain all the unsatisfied pairs $\mathcal{U}(\mathcal{A}, \mathcal{X})$. In particular, for each cycle induced in $c(G_{\mathcal{X} \cup \mathcal{U}(\mathcal{A},\mathcal{X})})$, at least one edge from that cycle cannot be included in $\hat{\mathcal{X}}$. Therefore, at least $c(G_{\mathcal{X} \cup \mathcal{U}(\mathcal{A},\mathcal{X})})$ elements of $\mathcal{U}(\mathcal{A}, \mathcal{X})$ cannot appear in $\hat{\mathcal{X}}$. The lemma follows. ■

This upper bound is tight and we give an algorithm that achieves this upper bound. The algorithm is described in Algorithm 1.

Algorithm 1: Restricted Block Ordering Problem with DCJ

Data: $\mathcal{A}$, circular-unichromosomal genome, $\mathcal{X}$ partial genome.
Result: $\hat{\mathcal{X}}$, circular-unichromosomal genome with $\mathcal{X} \subseteq \hat{\mathcal{X}}$ such that $d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}})$ is minimum.

begin
  $G_{\mathcal{X}} \longleftarrow$ partial genome graph;
  $\mathcal{M}(\mathcal{X}) \longleftarrow$ missing extremities;
  $AG_{\mathcal{A},\mathcal{X}} \longleftarrow$ partial adjacency graph;
  $\mathcal{U}(\mathcal{A}, \mathcal{X}) \longleftarrow$ unsatisfied pairs;
  $\hat{\mathcal{X}} \longleftarrow \mathcal{X}$;
  $\mathcal{E}(\hat{\mathcal{X}}) \longleftarrow$ excluded adjacencies;
  % main for loop;
  for $\{u, v\} \in \mathcal{U}(\mathcal{A}, \mathcal{X})$ **do**
    if $\{u, v\} \notin \mathcal{E}(\hat{\mathcal{X}})$ **then**
      let u' be such that $\{u, u'\} \in \mathcal{E}(\hat{\mathcal{X}})$;
      let v' be such that $\{v, v'\} \in \mathcal{E}(\hat{\mathcal{X}})$;
      $\mathcal{E}(\hat{\mathcal{X}}) \longleftarrow \mathcal{E}(\hat{\mathcal{X}}) \cup \{u', v'\} \setminus \{u, u'\} \setminus \{v, v'\}$;
      $\hat{\mathcal{X}} \longleftarrow \hat{\mathcal{X}} \cup \{u, v\}$;
      $\mathcal{M}(\mathcal{X}) \longleftarrow \mathcal{M}(\mathcal{X}) \setminus \{u, v\}$;
  while $\mathcal{M}(\mathcal{X}) \neq \emptyset$ **do**
    for $i, j \in \mathcal{M}(\mathcal{X})$ **do**
      if $\{i, j\} \notin \mathcal{E}(\hat{\mathcal{X}})$ **then**
        $\hat{\mathcal{X}} \longleftarrow \hat{\mathcal{X}} \cup \{i, j\}$;
        $\mathcal{M}(\mathcal{X}) \longleftarrow \mathcal{M}(\mathcal{X}) \setminus \{i, j\}$;
  Output $\hat{\mathcal{X}}$;
end

The running time is linear in the number of genes, N.

The algorithm achieves the upper bound given in Lemma 10 by greedily adding unsatisfied pairs to the partial genome $\mathcal{X}$ as long as they do not induce a cycle in the partial genome graph $G_{\hat{\mathcal{X}}}$. This is verified in the main for loop by checking whether an unsatisfied pair is also a member of the set of excluded adjacencies, i.e. the adjacencies whose addition to the partial genome $\hat{\mathcal{X}}$ would induce a cycle in the partial genome graph. The final set of adjacencies added to $\hat{\mathcal{X}}$ in the second loop connects all the remaining extremities in such a way that no excluded adjacencies are added to $\hat{\mathcal{X}}$ and one final cycle is added to the adjacency graph.

Theorem 10 (Restricted Completion)**.** *Given a circular-unichromosomal genome $\mathcal{A}$ and partial genome $\mathcal{X}$ on N genes, an optimal solution to the restricted block ordering problem $\hat{\mathcal{X}}$ of $\mathcal{X}$ will exhibit the DCJ distance:*

$$d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}}) = N - \left(c(AG_{\mathcal{A},\mathcal{X}}) + \frac{\mid \mathcal{M}(\mathcal{X}) \mid}{2} - c(G_{\mathcal{X} \cup \mathcal{U}(\mathcal{A},\mathcal{X})}) + 1 \right), \tag{4.7}$$

where $c(AG_{\mathcal{A},\mathcal{X}})$ is the number of cycles in the partial adjacency graph $AG_{\mathcal{A},\mathcal{X}}$ and $c(G_{\mathcal{X} \cup \mathcal{U}(\mathcal{A},\mathcal{X})})$ is the number of cycles in the genome graph for the complete genome $\mathcal{X} \cup \mathcal{U}(\mathcal{A}, \mathcal{X})$.

4.6 Future Directions

Traditionally, ESP reads are assumed to represent clones of a single cancer genome sequence. But current ESP sequencing technology uses approximately $1\,\mu$g physical DNA sample in order to generate and sequence clones. Even if the DNA extracted for use in an ESP experiment represents molecules from a single tissue sample from a single patient, there is no guarantee that all of the DNA comes from a single genome. In particular, humans contain diploid genomes, so clones could be made from either copy of a patient's genome. Moreover, because cancer is characterized by a progressive series of somatic mutations, a single tissue might contain many differently mutated versions of the cancer genome. As a result, it is possible that a set of ESP reads represent clones taken from multiple different cancer genomes even if they come from the same tissue sample. Therefore, it is reasonable to take this assumption into consideration when characterizing tumor rearrangements using ESP data.

Here we formalize the problem of inferring a set of differently mutated cancer genomes from a set of measured adjacencies. As in our description of the completion problem, we

assume that our measured data is incomplete, representing some set of breakpoints (i.e. adjacencies) that are known to exist in the cancer sample but we do not assume that the set of measured adjacencies from our unknown tumor sample is comprehensive.

Definition 33. *A k-**completion**, $\hat{\mathcal{X}}^k$, of $\mathcal{X}$ is a set of k different genomes such that $\mathcal{X} \subseteq \bigcup_{\hat{\mathcal{X}}^k}$.*

Again, let $\mathcal{A}$ denote a reference healthy genome. We represent a set $\hat{\mathcal{X}}^k$ of k cancer genomes that represent mutations of a healthy genome as a rooted tree $\mathcal{T}_{\hat{\mathcal{X}}^k}$ on $\hat{\mathcal{X}}^k \cup \{\mathcal{A}\}$, rooted at $\mathcal{A}$. We call $\mathcal{T}_{\hat{\mathcal{X}}^k}$ a *mixture tree*. Given a distance metric, such as DCJ, the total distance on a mixture tree $\mathcal{T}_{\hat{\mathcal{X}}^k} = (V, E)$ is given by:

$$d_{DCJ}(\mathcal{T}_{\hat{\mathcal{X}}^k}) = \sum_{(u,v)\in E} d_{DCJ}(u, v). \tag{4.8}$$

We now suggest the following parsimony-based problem.

Definition 34. *Given a set of measured adjacencies, $\mathcal{X}$, and an integer $k > 0$, the k-**Mixture Problem** is to find a k-completion such that $d_{DCJ}(\mathcal{T}_{\hat{\mathcal{X}}^k})$ is minimum.*

Again, we can distinguish between restricted and unrestricted versions of the problem. Note that when $k = 1$, the (un)restricted k-mixture problem is equivalent to the (un)restricted completion problem.

As a starting point, we consider here the k-mixture problem when $k = 2$. There are exactly two different (unlabeled) rooted tree topologies on 3 nodes, namely the tree comprised of a root and two daughter nodes, that we shall refer to as the *branch topology*, and the tree comprised of a root, one internal node, and one leaf, that we shall refer to as the *path topology*.

First, we provide a motivating example to show that both topologies must be considered when solving an instance of the k-mixture problem for $k = 2$.

Consider the example on $N = 4$ genes with reference genome and with measured adjacencies as follows:

$$\begin{aligned} \mathcal{A} &= \{\{a_h, b_t\}, \{b_h, c_t\}, \{c_h, d_t\}, \{d_h, a_t\}\}, \\ \mathcal{X} &= \{\{a_h, b_h\}, \{b_t, c_t\}, \{c_h, d_h\}, \{a_h, c_t\}, \{b_h, c_h\}, \{b_t, d_t\}\}. \end{aligned} \tag{4.9}$$

Suppose we are interested in the restricted version of the problem wherein the k-completion is required to be comprised of circular-unichromosomal genomes. In this example, there is

only one way to partition the set of adjacencies into two sets representing partial genomes (i.e. without repeated extremities), namely, the partition defined by the first three elements listed in $\mathcal{X}$ above and the second three elements listed in $\mathcal{X}$. That partition defines two partial genomes for which there exist unique completions. The resulting completions are, respectively, $\mathcal{B} = \{\{a_h, b_h\}, \{b_t, c_t\}, \{c_h, d_h\}, \{d_t, a_t\}\}$ and $\mathcal{C} = \{\{a_h, c_t\}, \{b_h, c_h\}, \{b_t, d_t\}, \{d_h, a_t\}\}$. The mixture tree on $\mathcal{B}$ and $\mathcal{C}$ that corresponds to the branch topology admits a total tree-distance of $d_{DCJ}\mathcal{T} = d_{DCJ}(\mathcal{A}, \mathcal{B}) + d_{DCJ}(\mathcal{A}, \mathcal{C}) = 2+2 = 4$. There are two possible labelings of the nodes in a mixture tree corresponding to the path topology. The first labeling admits a total tree-distance of $d_{DCJ}\mathcal{T} = d_{DCJ}(\mathcal{A}, \mathcal{B}) + d_{DCJ}(\mathcal{B}, \mathcal{C}) = 2 + 3 = 5$. The second labeling admits a total tree-distance of $d_{DCJ}\mathcal{T} = d_{DCJ}(\mathcal{A}, \mathcal{C}) + d_{DCJ}(\mathcal{C}, \mathcal{B}) = 2 + 3 = 5$ as well. Therefore, in this example, a tree with the branch topology admits a more parsimonious mixture than a tree with the path topology.

Conversely, consider the example with the same reference genome $\mathcal{A}$ and with measured adjacencies as follows:

$$\begin{aligned} \mathcal{A} &= \{\{a_h, b_t\}, \{b_h, c_t\}, \{c_h, d_t\}, \{d_h, a_t\}\} , \\ \mathcal{X} &= \{\{a_h, b_h\}, \{c_t, d_t\}, \{c_h, a_t\}, \{a_h, c_h\}, \{b_h, a_t\}, \{b_t, d_t\}\} . \end{aligned} \tag{4.10}$$

In this example, there are several different partitions of $\mathcal{X}$ that represent two partial genomes, and those partial genomes admit several different completions. However, an optimal mixture with a branch topology admits a total score of 5 whereas an optimal mixture with a path topology admits a total score of 4. Therefore, in this example, a tree with the path topology admits a more parsimonious mixture than a tree with the branch topology.

We hope to characterize the instances in which the two tree topologies for $k = 2$ admit solutions with different scores. Given a set of measured adjacencies, and a tree topology, then, we hope to devise an efficient algorithm to construct the optimal k-completion of $\mathcal{X}$. We expect that the algorithms given in the previous section will be the basis for any algorithms we devise to solve the k-mixture problem.

As a first step toward characterizing the inputs to the 2-mixture problem for which one of the two possible tree topologies admits a more parsimonious solution than the other, we note that for a reference genome $\mathcal{A}$, a set of adjacencies $\mathcal{X}$, and a 2-completion comprised of genomes $\mathcal{B}$ and $\mathcal{C}$, of $\mathcal{X}$, we can directly determine the best topology if we know the pairwise distances between $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, respectively. In particular, the total distance on a tree with a branch topology is equal to $d_{DCJ}(\mathcal{A}, \mathcal{B}) + d_{DCJ}(\mathcal{B}, \mathcal{C})$. By contrast, the distance on a tree with path topology is equal to either $d_{DCJ}(\mathcal{A}, \mathcal{B}) + d_{DCJ}(\mathcal{B}, \mathcal{C})$ or $d_{DCJ}(\mathcal{A}, \mathcal{C}) +$

$d_{DCJ}(\mathcal{C}, \mathcal{B})$, depending on the placement of genomes at the nodes in the tree. This gives us the following lemma.

Lemma 11. *Given a circular-unichromosomal genome $\mathcal{A}$ on N genes and a set of adjacencies $\mathcal{X}$ in which each extremity appears no more than twice, let genomes $\mathcal{B}$ and $\mathcal{C}$ be a 2-completion of $\mathcal{X}$. If $d_{DCJ}(\mathcal{B}, \mathcal{C}) < d_{DCJ}(\mathcal{A}, \mathcal{C})$ and $d_{DCJ}(\mathcal{B}, \mathcal{C}) < d_{DCJ}(\mathcal{A}, \mathcal{B})$, then a path topology is more parsimonious than a branch topology. If $d_{DCJ}(\mathcal{A}, \mathcal{B}) < d_{DCJ}(\mathcal{B}, \mathcal{C})$ and $d_{DCJ}(\mathcal{A}, \mathcal{C}) < d_{DCJ}(\mathcal{B}, \mathcal{C})$, then a branch topology is more parsimonious than a path topology.*

Therefore, given a pair of partial genomes, we can construct a 2-completion in linear time using one of the approaches described in Sections 4.5.1 and 4.5.2. And given a 2-completion of a set of adjacencies, we can decide which tree topology admits a more parsimonious solution in linear time. However, it remains an open question as to whether the there exists an efficient method for finding a partition of a set of adjacencies $\mathcal{X}$ into two partial genomes whose completions will admit an optimal mixture tree.

In some cases, we find that a single genome completion of a set of adjacencies will be as parsimonious as any k-completion for $k > 1$, indicating that the measured adjacencies were most likely taken from a single tumor genome instead of from a mixture. We note that if a set of adjacencies contains some extremity more than once then it is not possible to construct a single genome that contains all the adjacencies represented. But in the case that a set of adjacencies $\mathcal{X}$ contains each extremity at most once, we can show that, in the unrestricted case, any 2-completion of $\mathcal{X}$ on a tree with branch topology is no better than a (single genome) completion.

Lemma 12. *Given a circular-unichromosomal genome $\mathcal{A}$ on N genes and a set of adjacencies $\mathcal{X}$ in which each extremity in $\mathcal{A}$ appears no more than once, let $\mathcal{G} = \{\mathcal{X}_1, \mathcal{X}_2\}$ be a 2-completion of $\mathcal{X}$ with branch topology and let $\hat{\mathcal{X}}$ be an optimal, unrestricted completion of $\mathcal{X}$. Then, $d_{DCJ}(\mathcal{T}_G) \geq d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}})$.*

Proof: First, we note that $\mathcal{X}_1$ and $\mathcal{X}_2$ partition the adjacencies in $\mathcal{X}$, so $\mid \mathcal{X} \mid = \mid \mathcal{X}_1 \mid + \mid \mathcal{X}_2 \mid$. Moreover, for a partition $\{\mathcal{X}_1, \mathcal{X}_2\}$ of $\mathcal{X}$, the total number of cycles in their respective partial adjacency graphs cannot exceed the number of cycles in the partial adjacency graph for $\mathcal{X}$, i.e. $c(AG_{\mathcal{A},\mathcal{X}}) \geq c(AG_{\mathcal{A},\mathcal{X}_1}) + c(AG_{\mathcal{A},\mathcal{X}_2})$. This is because the set of cycles in $AG_{\mathcal{A},\mathcal{X}_1}$ and the set of cycles in $AG_{\mathcal{A},\mathcal{X}_2}$ are both disjoint subsets of the set of cycles in $AG_{\mathcal{A},\mathcal{X}}$.

Now, in order to show the lemma, we must show that the total distance $d_{DCJ}(\mathcal{A}, \mathcal{X}_1) +$

$d_{DCJ}(\mathcal{A}, \mathcal{X}_2)$ is at least as much as the distance $d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}})$. By Thm 8, we have that $d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}}) =\mid \mathcal{X} \mid -c(AG_{\mathcal{A},\mathcal{X}})$. Thus, we can show the following bound.

$$\begin{aligned} d_{DCJ}(\mathcal{A}, \hat{\mathcal{X}}) &\leq d_{DCJ}(\mathcal{A}, \mathcal{X}_1) + d_{DCJ}(\mathcal{A}, \mathcal{X}_2), \\ \mid \mathcal{X} \mid -c(AG_{\mathcal{A},\mathcal{X}}) &\leq \mid \mathcal{X}_1 \mid -c(AG_{\mathcal{A},\mathcal{X}_1}) + \mid \mathcal{X}_2 \mid -c(AG_{\mathcal{A},\mathcal{X}_2}), \\ \mid \mathcal{X} \mid -c(AG_{\mathcal{A},\mathcal{X}}) &\leq \mid \mathcal{X} \mid -\left(c(AG_{\mathcal{A},\mathcal{X}_1}) + c(AG_{\mathcal{A},\mathcal{X}_2})\right), \\ c(AG_{\mathcal{A},\mathcal{X}}) &\geq c(AG_{\mathcal{A},\mathcal{X}_1}) + c(AG_{\mathcal{A},\mathcal{X}_2}). \end{aligned}$$

■

We can extend this proof to any set of adjacencies in which each extremity appears no more than d adjacencies; given such a set of adjacencies, an unrestricted d-completion with "branch topology" (i.e. a star graph with d daughter nodes) will admit at least as parsimonious a solution with respect to total DCJ distance on edges as any $d+1$ completion with branch topology.

We believe that the techniques we introduced in this chapter for addressing the block ordering problem with respect to DCJ distance using the adjacency graph data structure is an intuitive and simple framework. We conjecture that it may be possible to extend this framework to address the k-mixture problem with respect to DCJ distance.

The k-mixture problem, however, might also prove interesting if we consider different measures of parsimony, such as reversal distance. As we pointed out at the beginning of this chapter, DCJ distance is a pretty good approximation for reversal distance between a pair of signed genomes. However, there are examples for which DCJ distance is strictly less than reversal distance. Due to this discrepancy, we cannot merely extrapolate the result from Lemma 12 to a similar problem where the measure of parsimony is reversal distance. Consider the following example.

Suppose that the reference genome $\mathcal{A}$ is now linear-unichromosomal and is the identity permutation on 6 genes. Let $\mathcal{X} = \{\{1_h, 4_h\}, \{2_t, 5_t\}, \{3_h, 6_h\}, \{4_t\}\}$. (Note that $\mathcal{X}$ contains a telomere.) The genome $\hat{\mathcal{X}} = -5{+}2{+}3{-}6{+}1{-}4$ is the completion (represented now as a signed string instead of as a set of adjacencies and telomeres) that minimizes the reversal distance to $\mathcal{A}$: $d_{rev}(\mathcal{A}, \hat{\mathcal{X}}) = 3$. For $k = 2$, a most-parsimonious k-completion on a tree with branch topology is $\mathcal{G} = \{\mathcal{B} = (+1\,{-}4\,{-}3\,{-}2\,{+}5\,{+}6), \mathcal{C} = (+1\,{+}2\,{+}3\,{-}6\,{-}5\,{-}4)\}$. The optimal mixture tree with branch topology for the linear-unichromosomal case with respect to reversal distance is given in Fig. 4.7. The total number of reversals on the tree is

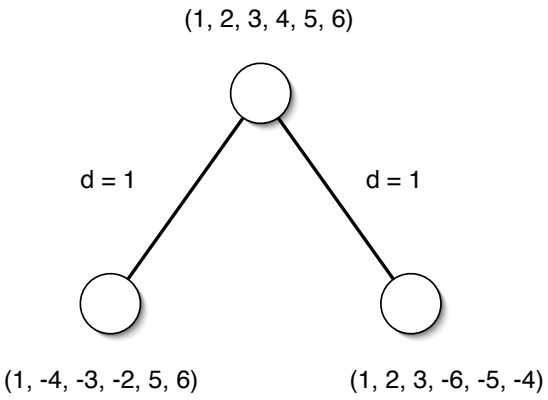

Figure 4.7: A mixture tree $\mathcal{T}$ on genomes $\mathcal{B} = (+1, -4, -3, -2, +5, +6)$ and $\mathcal{C} = (+1, +2, +3, -6, -5, -4)$. The distances from the root to each of its children are: $d(\mathcal{A}, \mathcal{B}) = d(\mathcal{A}, \mathcal{C}) = 1$, and thus $d_{rev}(\mathcal{T}) = 2$.

only 2, which is less than the total number of reversals between $\mathcal{A}$ and the optimal (single genome) completion $\hat{\mathcal{X}}$.

The k-completion problem seems related to the well-studied problem of constructing a phylogenetic tree to represent a rearrangement history for a set of known genomes of common ancestry. For example, [14] and [22] consider the problem of computing a phylogenetic tree for a set of known genomes by minimizing the total *breakpoint distance* on the tree, and [16] consider the a similar problem but minimize the total reversal distance on the tree. However, in the phylogenetic tree problem, the leaves of the tree are a set of known genomes and the goal is to compute a set of unknown ancestral genomes that represent internal nodes in the tree. We, instead, are interested in constructing the genomes at all the nodes in the tree from impartial data.

For a set of adjacencies and telomeres $\mathcal{X}$ and an integer k, finding a most-parsimonious mixture of k of cancer genomes amounts to partitioning $\mathcal{X}$ into k sets such that we may construct k genomes, each containing some subset of the elements in $\mathcal{X}$. There are exponentially many ways to do this; in particular, there are $\sum_{i=1}^{k} S(\mid \mathcal{X} \mid, i)$ different ways, where $S(n, k)$ is the Stirling number of the second kind. Then given an integer k, there are $(k+1)^{k-1}$ different possible labeled trees on $k+1$ nodes and thus as many mixture trees on k permutations. Thus, an exhaustive search procedure could not find an optimal mixture tree efficiently.

BIBLIOGRAPHY

[1] Max A. Alekseyev and Pavel A. Pevzner. Whole genome duplications and contracted breakpoint graphs. *SICOMP*, 36(6):1748–1763, 2007.

[2] C. Alkan, J. M. Kidd, T. Marques-Bonet, G. Aksay, F. Antonacci, F. Hormozdiari, J. O. Kitzman, C. Baker, M. Malig, O. Mutlu, S. C. Sahinalp, R. A. Gibbs, and E. E. Eichler. Personalized copy number and segmental duplication maps using next-generation sequencing. *Nat. Genet.*, 41:1061–1067, 2009.

[3] Can Alkan, Evan E. Eichler, Jeffrey A. Bailey, S. Cenk Sahinalp, and Eray Tuzun. The role of unequal crossover in alpha-satellite DNA evolution: a computational analysis. *J Comp Biol*, 11(5):933–944, 2004.

[4] David A. Bader, Bernard M.E. Moret, and Mi Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8(5):483–491, 2001.

[5] V. Bafna and P.A. Pevzner. Genome rearrangements and sorting by reversals. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:148–157, 1993.

[6] J.A. Bailey and E.E. Eichler. Primate segmental duplications: crucibles of evolution, diversity and disease. *Nat. Rev. Genet.*, 7:552–564, 2006.

[7] J.A. Bailey, Z. Gu, R.A. Clark, K. Reinert, R.V. Samonte, S. Schwartz, M.D. Adams, E.W. Myers, P.W. Li, and E.E. Eichler. Recent segmental duplications in the human genome. *Science*, 297:1003–1007, 2002.

[8] Ali Bashir, Chun Ye, Alkes L. Price, and Vineet Bafna. Orthologous repeats and mammalian phylogenetic inference. *Genome Research*, 15(7):998–1006, 2005.

[9] Anne Bergeron, Julia Mixtacki, and Jens Stoye. A unifying view of genome rearrangements. *Algorithms in Bioinformatics*, pages 163–173, 2006.

[10] Piotr Berman and Sridhar Hannenhalli. Fast sorting by reversal. In *Proc. 7th Ann. Symposium on Combinatorial Pattern Matching*, pages 168–185, 1996.

[11] Denis Bertrand, Mathieu Lajoie, and Nadia El-Mabrouk. Inferring ancestral gene orders for a family of tandemly arrayed genes. *J Comp Biol*, 15(8):1063–1077, 2008.

[12] M. Blanchette, G. Bourque, and D. Sankoff. Breakpoint phylogenies. In *Genome Informatics Workshop (GIW)*, volume 8, pages 25–34, 1997.

[13] M. Blanchette, E.D. Green, W. Miller, and D. Haussler. Reconstructing large regions of an ancestral mammalian genome in silico. *Genome Res.*, 14:2412–2423, 2004.

[14] Mathieu Blanchette, Takashi Kunisawa, and David Sankoff. Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution*, 49(2):193–203, 1999.

[15] R. Blekhman, A. Oshlack, and Y. Gilad. Segmental duplications contribute to gene expression differences between humans and chimpanzees. *Genetics*, 182:627–630, 2009.

[16] Guillaume Bourque and Pavel A. Pevzner. Genome-scale evolution: reconstructing gene orders in the ancestral species. *Gen. Res.*, 12:26–36, 2002.

[17] Alberto Caprara. Formulations and hardness of multiple sorting by reversals. In *RECOMB '99: Proceedings of the third annual international conference on Computational molecular biology*, pages 84–93, New York, NY, USA, 1999. ACM.

[18] Kamalika Chaudhuri, Kevin Chen, Radu Mihaescu, and Satish Rao. On the tandem duplication-random loss model of genome rearrangement. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 564–570, New York, NY, USA, 2006. ACM.

[19] Xin Chen, Jie Zheng, Zheng Fu, Peng Nan, Yang Zhong, Stefano Lonardi, and Tao Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Trans. Comp. Biol. Bioinformatics*, 2(4):302–315, 2005.

[20] Zhi-Zhong Chen, Lusheng Wang, and Zhanyong Wang. Approximation algorithms for reconstructing theduplication history oftandemrepeats. *Algorithmica*, pages 1748–1763, 2008.

[21] F.D. Ciccarelli, C. von Mering, M. Suyama, E.D. Harrington, E. Izaurralde, and P. Bork. Complex genomic rearrangements lead to novel primate gene function. *Genome Res.*, 15:343–351, 2005.

[22] M.E. Cosner, R.K. Jansen, B.M.E. Moret, L.A. Raubeson, L.S. Wang, T. Warnow, and S. Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in campanulaceae. *Comparative Genomics (DCAF 2000)*, pages 104–115, 2000.

[23] Nadia El-Mabrouk. Genome rearrangement by reversals and insertions/deletions of contiguous segments. In *In Proc. 11th Ann. Symp. Combin. Pattern Matching (CPM00)*, volume 1848, pages 222–234, Berlin, 2000. Springer-Verlag.

[24] Nadia El-Mabrouk. Reconstructing an ancestral genome using minimum segments duplications and reversals. *J. Comput. Syst. Sci.*, 65(3):442–464, 2002.

[25] Nadia El-Mabrouk and David Sankoff. The reconstruction of doubled genomes. *SICOMP*, 32(3):754–792, 2003.

[26] Olivier Elemento, Olivier Gascuel, and Marie-Paule Lefranc. Reconstructing the duplication history of tandemly repeated genes. *Mol Biol Evol*, 19(3):278–288, 2002.

[27] Funda Ergun, S. Muthukrishnan, and S. Cenk Sahinalp. Comparing sequences with segment rearrangements. In *In Proceedings of Foundations of Software Technology and Theoretical Computer Science*, pages 183–194. Springer, 2003.

[28] Éric Gaul and Mathieu Blanchette. Ordering partially assembled genomes using gene arrangements. In Guillaume Bourque and Nadia El-Mabrouk, editors, *Comparative Genomics*, volume 4205 of *Lecture Notes in Computer Science*, pages 113–128. Springer, 2006.

[29] Paolo Giudici and Robert Castelo. Improving markov chain monte carlo model search for data mining. *Machine Learning*, 50(1-2):127–158, 2003.

[30] Sridhar Hannenhalli and Pavel Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. In *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 178–189, New York, NY, USA, 1995. ACM.

[31] J.E. Horvath, C.L. Gulden, R.U. Vallente, M.Y. Eichler, M. Ventura, J.D. McPherson, T.A. Graves, R.K. Wilson, S. Schwartz, M. Rocchi, and E.E. Eichler. Punctuated duplication seeding events during the evolution of human chromosome 2p11. *Genome Res.*, 15:914–927, 2005.

[32] Zhaoshi Jiang, Haixu Tang, Mario Ventura, Maria Francesca Cardone, Tomas Marques-Bonet, Xinwei She, Pavel A Pevzner, and Evan E Eichler. Ancestral reconstruction of segmental duplications reveals punctuated cores of human genome evolution. *Nature Genetics*, 39:1361–1368, 2007.

[33] Crystal Kahn, Shay Mozes, and Benjamin Raphael. Efficient algorithms for analyzing segmental duplications with deletions and inversions in genomes. *Algorithms for Molecular Biology*, 5(1):11, 2010.

[34] Crystal L. Kahn, Borislav H. Hristov, and Benjamin J. Raphael. Parsimony and likelihood reconstruction of human segmental duplications. *Bioinformatics*, 26(18):i446–i452, 2010.

[35] Crystal L. Kahn, Shay Mozes, and Ben J. Raphael. Efficient algorithms for analyzing segmental duplications, deletions, and inversions in genomes. In *WABI*, volume 5724 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2009.

[36] Crystal L. Kahn and Ben J. Raphael. Analysis of segmental duplications via duplication distance. *Bioinformatics*, 24:i133–138, 2008.

[37] Crystal L. Kahn and Ben J. Raphael. A parsimony approach to analysis of human segmental duplications. In *Pacific Symposium on Biocomputing*, volume 14, pages 126–137, 2009.

[38] P. M. Kim, H. Y. Lam, A. E. Urban, J. O. Korbel, J. Affourtit, F. Grubert, X. Chen, S. Weissman, M. Snyder, and M. B. Gerstein. Analysis of copy number variants and segmental duplications in the human genome: Evidence for a change in the process of formation in recent evolutionary history. *Genome Res.*, 18:1865–1874, 2008.

[39] Mathieu Lajoie, Denis Bertrand, Nadia El-Mabrouk, and Oliver Gascuel. Duplication and inversion history of a tandemly repeated gene family. *J. Comp. Bio.*, 14(4):462–478, 2007.

[40] E.V. Linardopoulou, S.S. Parghi, C. Friedman, G.E. Osborn, S.M. Parkhurst, and B.J. Trask. Human subtelomeric WASH genes encode a new subclass of the WASP family. *PLoS Genet.*, 3:e237, 2007.

[41] T. Liu, B.M.E. Moret, and D.A. Bader. An exact linear-time algorithm for computing genomic distances under inversions and deletions, 2003.

[42] J.R. Lupski and P. Stankiewicz. Genomic disorders: molecular mechanisms for rearrangements and conveyed phenotypes. *PLoS Genet.*, 1:e49, 2005.

[43] Mark Marron, Krister M. Swenson, and Bernard M. E. Moret. Genomic distances under deletions and insertions. *TCS*, 325(3):347–360, 2004.

[44] J S McCaskill. The equilibrium partition function and base pair binding probabilities for rna secondary structure. *Biopolymers*, 29(6-7):1105–19.

[45] Aleksandar Milosavljevic, David Haussler, and Jerzy Jurka. Informed parsimonious inference of prototypical genetic sequences. In *COLT '89: Proceedings of the second annual workshop on Computational learning theory*, pages 102–117, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[46] Bernard Moret, Bernard M. E, Stacia Wyman, David A. Bader, Tandy Warnow, and Mi Yan. A new implementation and detailed study of breakpoint analysis, 2001.

[47] Bernard M.E. Moret, Li-San Wang, Tandy Warnow, and Stacia K. Wyman. New approaches for reconstructing phylogenies from gene order data. *Bioinformatics*, 17(suppl_1):S165–173, 2001.

[48] Sean O'Rourke, Noah Zaitlen, Nebojsa Jojic, and Eleazar Eskin. Reconstructing the phylogeny of mobile elements. In *In Proceedings of the Eleventh Annual Conference on Research in Computational Biology (RECOMB 2007)*, pages 196–210, Berlin, 2007. Springer.

[49] I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. In *Elec. Colloq. on Comput. Complexity*, volume 71, 1998.

[50] Pavel Pevzner. *Computational molecular biology : an algorithmic approach*. MIT Press, Cambridge, Mass., 2000.

[51] E. D. Pleasance, R. K. Cheetham, P. J. Stephens, D. J. McBride, S. J. Humphray, C. D. Greenman, I. Varela, M.-L. Lin, G. R. Ordóñez, G. R. Bignell, K. Ye, J. Alipaz, M. J. Bauer, D. Beare, A. Butler, R. J. Carter, L. Chen, A. J. Cox, S. Edkins, P. I. Kokko-Gonzales, N. A. Gormley, R. J. Grocock, C. D. Haudenschild, M. M. Hims, T. James, M. Jia, Z. Kingsbury, C. Leroy, J. Marshall, A. Menzies, L. J. Mudie, Z. Ning, T. Royce, O. B. Schulz-Trieglaff, A. Spiridou, L. A. Stebbings, L. Szajkowski, J. Teague, D. Williamson, L. Chin, M. T. Ross, P. J. Campbell, D. R. Bentley,

P. A. Futreal, and M. R. Stratton. A comprehensive catalogue of somatic mutations from a human cancer genome. *Nature*, 463:191–196, January 2010.

[52] A. L. Price, E. Eskin, and P. A. Pevzner. Whole-genome analysis of Alu repeat elements reveals complex evolutionary history. *Genome Res.*, 14:2245–2252, 2004.

[53] D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B.F. Lang, and R. Cedergren. Gene order comparisons for phylogenetic inference: evolution of the mitochondrial genome. *Proc. Natl. Acad. Sci. U.S.A.*, 89(14):6575–6579, 1992.

[54] David Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.

[55] David Sankoff and Mathieu Blanchette. The median problem for breakpoints in comparative genomics. In *COCOON '97: Proceedings of the Third Annual International Conference on Computing and Combinatorics*, pages 251–264, London, UK, 1997. Springer-Verlag.

[56] Krister M. Swenson, Mark Marron, Joel V. Earnest-Deyoung, and Bernard M. E. Moret. Approximating the true evolutionary distance between two genomes. *J. Exp. Algorithmics*, 12:1–17, 2008.

[57] K. Vandepoele, N. Van Roy, K. Staes, F. Speleman, and F. van Roy. A novel gene family NBPF: intricate structure generated by gene duplications during primate evolution. *Mol. Biol. Evol.*, 22:2265–2274, 2005.

[58] G. A. Watterson, W. J. Ewens, T. E. Hall, and A. Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99(1):1 – 7, 1982.

[59] Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

[60] Yu Zhang, Giltae Song, Tomás Vinar, Eric D. Green, Adam C. Siepel, and Webb Miller. Reconstructing the evolutionary history of complex human gene clusters. In *Proc. 12th Int'l Conf. on Research in Computational Molecular Biology (RECOMB)*, pages 29–49, 2008.

[61] M. C. Zody, Z. Jiang, H. C. Fung, F. Antonacci, L. W. Hillier, M. F. Cardone, T. A. Graves, J. M. Kidd, Z. Cheng, A. Abouelleil, L. Chen, J. Wallis, J. Glasscock, R. K. Wilson, A. D. Reily, J. Duckworth, M. Ventura, J. Hardy, W. C. Warren, and E. E.

Eichler. Evolutionary toggling of the MAPT 17q21.31 inversion region. *Nat. Genet.*, 40:1076–1083, 2008.

APPENDIX A: A DISCUSSION OF ANCESTRAL GENOME RECONSTRUCTION USING DUPLICATIONS AND REVERSALS

The problem of computing a minimum number of duplication transposition and reversal operations needed to transform an ancestral genome (with no duplicates) into a present-day genome (with duplicates) has been studied by El-Mabrouk in [24]. To simplify the problem, El-Mabrouk [24] devises a model of evolution in which she gives a method for solving the general problem of computing minimum reversal/duplication transposition distance by solving an exponential number of polynomially solvable subproblems.

In particular, the subproblem considered in [24] is that of computing the minimum reversal/duplication transposition distance between a *non-ambiguous* ancestral genome H (that contains every character at most once) and a *semi-ambiguous* genome G (in which no character appears more than twice). It is shown that the minimum number of reversals and duplication transpositions needed to transform H into G can be computed by assuming the rearrangements occur in two distinct phases. First, the non-ambiguous H undergoes a series of duplication transpositions, transforming it into an intermediate ancestor I having the same character composition as G, but not necessarily in the same order. Then, the intermediate ancestor I evolves through a series of reversal operations, resulting in the present-day genome G.

The problem of finding the reversal/duplication transposition distance between a semi-ambiguous genome G and a non-ambiguous genome H then is merely the problem of reconstructing a semi-ambiguous intermediate ancestor I that minimizes the sum of the duplication transposition distance between any non-ambiguous ancestor H and I and the

reversal distance between I and G.

In [24], the problem of computing the reversal/duplication transposition distance between a non-ambiguous ancestral genome and a present-day semi-ambiguous genome G, $RD(G)$, is formalized as

$$RD(G) = \min_{I} [R(G, I) + D(I)] \tag{A.1}$$

where $R(G, I)$ is the reversal distance between G and I, $D(I)$ is defined as the number of maximal, repeated substrings of I, and the minimum is taken over all possible permutations I of the set of characters comprising G.

The algorithm given in [24], therefore, implies that the minimum number of duplication transpositions needed to transform a non-ambiguous H into a semi-ambiguous I is equal to the number of maximal repeated substrings of I. However, this simplification is incorrect.

Claim 7. *Let H be a non-ambiguous string and let I be a semi-ambiguous string that evolved from H through a series of duplication transpositions. $D(I)$, the number of maximal, repeated substrings of I, is not necessarily equal to the minimum number of duplication transposition operations needed to transform H into I.*

Proof: We provide the following counterexample. Consider the strings:

$$H = abcdefg$$
$$I = ab\mathbf{de}cd\mathbf{bc}efg$$

In this example, the maximal repeated substrings of I are $\{d, e, b, c\}$, so $D(I) = 4$. However, a sequence of three duplication transposition operations would suffice to create I: first, duplicate bc in one operation, then duplicate d and e in two separate operations.

■

Note that in the example above, the duplication distance, $d(H, I)$ is two. (See Chapter 2 for a description of duplication distance.) The only cases considered in [24] are cases in which the ancestor genome H is non-ambiguous and the intermediate ancestor I is semi-ambiguous. In these cases, the duplication distance is no greater than the duplication transposition distance.

Claim 8. *If Y is semi-ambiguous and X is non-ambiguous, X is a subsequence of Y, and $\{Y\} \setminus \{X\} = \emptyset$ then $d(X, Y) \leq DT(X, Y)$, where $DT(X, Y)$ equals the number of duplication transpositions needed to transform X into Y and $\{X\}$ denotes the multiset of characters that appear in Y (and similar for $\{X\}$).*

Proof: Because Y is semi-ambiguous, no character of X is copied more than once in any sequence of duplication transposition or duplicate operations building Y from X. Therefore, all the strings that are copied in a minimum sequence of either duplication transposition or duplicate operations are substrings of the original input string X.

Consider a minimum-length sequence of duplication transpositions that transforms X into Y. Because all the duplication transpositions in such a sequence copy substrings of the original string X, there are corresponding duplicate operations, copying the same sequence of substrings of X, giving rise to Y. However, as we noted above, the optimal sequence of duplicate operations transforming X into Y may have strictly smaller length than that of the optimal sequence of duplication transposition operations yielding the same Y. Thus, $d(X, Y) \leq DT(X, Y)$. ■

It follows that duplication distance may provide a simpler explanation than duplication transposition distance in this subproblem of computing the minimum reversal/duplication transposition distance between a non-ambiguous string and a semi-ambiguous string. Moreover, the computation implied in [24] for retrieving the ancestor genome H from the intermediate, semi-ambiguous genome, I, is to delete exactly one copy of every maximal repeated substring of I. However, this may not yield an ancestor genome H such that $DT(H, I)$ is minimal. Consider, the example given in the proof of Claim 7. If we had deleted the not-bolded copies of each of the repeated characters, we would construct the ancestor genome $H' = adebcfg$. In this example, the duplication transposition distance $DT(H', I)$ is equal to four, the number of repeats. However, H' is not the ancestor genome that minimizes the duplication transposition distance. Therefore, the algorithm in [24] not only incorrectly computes the optimal value of the duplication transposition distance between I and some non-ambiguous H, it might also output an ancestor genome that is non-optimal.

Now we return to the reversal/duplication transposition distance problem posed in [24]. Under the assumption that a semi-ambiguous genome G evolved from a non-ambiguous genome H by a series of duplication transpositions followed by a series of reversals, the formulation of Equation A.1 can be corrected if we change the definition of $D(I)$ so that it no longer equals the number of repeats in I. Instead, let

$$D(I) = \min_{H \in \mathcal{A}} DT(H, I) \tag{A.2}$$

where $\mathcal{A}$ is defined as the set of all non-ambiguous subsequences of I that are obtained by

deleting exactly one copy of every duplicated gene. Thus, if I has d duplicated genes, $\mathcal{A}$ will contain 2^d elements.

Unfortunately, there is no known algorithm for computing a minimum sequence of duplication transpositions needed to transform a non-ambiguous genome into one with duplicates. The problem's difficulty results from the fact that the set of substrings that can be duplicated after the k^{th} duplication transposition operation depends on the first k operations. On the other hand, Claim 8 states that the duplication distance $d(H, I)$ for any $H \in \mathcal{A}$ is a lower bound for the duplication transposition distance.

APPENDIX B: ANOTHER PROBABILISTIC MODEL OF SEGMENTAL DUPLICATIONS

Here we present an alternative generative probabilistic model of segmental duplications that could be used in place of the probabilistic model presented in Section 3.2.1 to compute a maximum likelihood evolutionary history DAG. Given a source string X and an integer $k > 0$, the model here considers the weighted ensemble of all sequences of k duplicate operations that could result in the construction of some target string. Here the order of the duplicate operations is important; unlike in the model presented in Section 3.2.1, here we distinguish sequences of duplicate operations in which the substrings of X are duplicated in different orders even if the resulting target strings are identical. Then the likelihood of a particular target string Y, given the source X and integer k, depends on the weighted ensemble of all sequences of k duplicate operations that could be used to generate Y from X. In principle, this model explicitly considers all possible duplication scenarios and is, therefore, a generative model. Ultimately, we did not implement this model because the recurrence relation–although polynomial-time–was prohibitively slow.

Again, we assume that duplication blocks, represented as signed strings on an alphabet of duplicons, are built up from other duplication blocks through successive rounds of duplicate operations (see Def. 4). Recall the following definition.

Definition 17. *Given a source string X, a **generator** $\Psi_X = (X_{i_1,j_1}, \ldots, X_{i_k,j_k})$ is a sequence of substrings of X.*

Here, we redefine what it means for a generator to be *feasible* for a particular target string. As before, we define a generator to be feasible for a target string if the constituent substrings partition the target into mutually nonoverlapping subsequences. However, we now require the order of substrings that comprise the generator to correspond to an order in which they

$$
\begin{aligned}
X &= && abcde \\
Y^0 &= && \emptyset \\
Y^1 &= Y^0 \circ \delta_X(1,3,1) = && abc \\
Y^2 &= Y^1 \circ \delta_X(4,5,1) = && deabc \\
Y &= Y^2 \circ \delta_X(4,5,5) = && deabdec
\end{aligned}
$$

Figure B.1: An example of a sequence of duplicate operations that constructs $Y = deabdec$ from $X = abcde$. A feasible generator for Y is: $\Psi_X = (X_{1,3}, X_{4,5}, X_{4,5}) = ((abc), (de), (de))$.

may have been duplicated from the source to build the given target. In particular, we have the following redefinition.

Definition 35. *A generator $\Psi_X = (X_{i_1,j_1}, \ldots, X_{i_k,j_k})$ is **feasible** for a target string Y, that we denote as $\Psi_X \dashv Y$, if there exists a sequence of indices $(t_1, \ldots, t_k)$ such that $Y = \emptyset \circ \delta_X(i_1, j_1, t_1) \circ \cdots \circ \delta_X(i_k, j_k, t_k)$. (See Fig. B.1).*

For a given source string X and positive integer k we consider the space of all length-k generators Ψ_X. Again, we define a probability distribution on the collection of generators and we compute the partition function $Z_X^{(k)}$ of the weighted ensemble of all possible length-k generators. We define the event F as before: it is the event of choosing a length-k generator that is feasible for Y from the space of all length-k generators. Thusly, we define a probabilistic model that, given a target string Y, assigns a probability to F:

$$Pr[F \mid Y, X, k] = \frac{1}{Z_X^{(k)}} \sum_{\Psi_X \dashv Y : |\Psi_X| = k} \omega(\Psi_X)\,, \tag{B.1}$$

where $\mid \Psi_X \mid$ denotes the length of the generator and $\omega(\Psi_X)$ is the weight assigned to a generator. We assume the weight function has the same properties as in Section 3.2.2.

First, we review the algorithm to compute the partition function $Z_X^{(k)}$. Because we have not changed the definition of a generator, the partition function of the ensemble of all length-k generators can be computed as before. Every length-k generator whose elements have lengths that sum to l are scored the same (according to $\sigma(k, l)$), we can count the total number of such generators and then multiply by the score function. Again, let $C_X^{(k)}(l)$ equal the number of distinct length-k generators for which the sum of the lengths of the elements equals l. Recall that we gave an $O(\mid X \mid k)$-time algorithm for computing $C_X^{(k)}(l)$ in Lemma 5:

Lemma 5. *Let $X = x_1 \ldots x_{|X|}$ be a source string and let k and l be positive integers. The*

function $\mathcal{C}_X^{(k)}(l)$ satisfies the following recurrence.

$$\begin{aligned}\mathcal{C}_X^{(1)}(l) &= \mid X \mid -l+1, \\ \mathcal{C}_X^{(k)}(l) &= \sum_{l'=l-|X|}^{l-1} \mathcal{C}_X^{(k-1)}(l') \cdot (\mid X \mid -(l-l')+1).\end{aligned}$$

For a source string X and integers k, l, if we are given $\mathcal{C}_X^{(k)}(l)$, we can compute $Z_X^{(k)}$ efficiently by summing $\mathcal{C}_X^{(k)}(l)$ over all relevant lengths l, weighting each feasible generator appropriately according to the function $\sigma(k,l)$. Therefore, again we have the theorem:

Theorem 5. *Let $X = x_1 \ldots x_{|X|}$ be a source string and k be a positive integer. The partition function $Z_X^{(k)}$ satisfies the following.*

$$Z_X^{(k)} = \sum_{l=k}^{|X|\cdot k} \mathcal{C}_X^{(k)}(l) \cdot \sigma(k,l).$$

The recurrence in Lemma 5 can be computed in $O(\mid X \mid k)$ time, so $Z_X^{(k)}$ can be computed in $O(\mid X \mid^2 k^2)$ time according to Theorem 5.

Therefore, using the new probabilistic model, we can compute the partition function of length-k generators for a given source string just as we did in Section 3.2.1.

However, the new definition of a feasible set requires that we augment our recurrence for computing the restricted partition function $Q^{(k)}$ of feasible sets. Fortunately, there is an easy extension we can make to do this. Since all length-k generators that are feasible for a target string Y have lengths that sum to $\mid Y \mid$, we can score them all according to $\sigma(k, \mid Y \mid)$. We describe here a recurrence to compute the number of distinct length-k generators Ψ_X that are feasible for a given string Y.

Lemma 13. *Given a source string $X = x_1 \ldots x_{|X|}$ and a target string $Y = y_1, \ldots, y_{|Y|}$, the number $\eta_X^{(k)}(Y)$ of distinct length-k generators Ψ_X that are feasible for Y satisfies the*

following recurrence.

$$
\begin{aligned}
\eta_X^{(k)}(Y) &= \sum_{i:x_i=y_1} \eta_X^{(k)}(Y,i), \\
\eta_X^{(k)}(Y,i) &= \sum_{d=1}^{k} \eta_X^{(k)}(Y,i,d), \\
\eta_X^{(1)}(Y,i,d) &= \begin{cases} 1 & \textit{if } Y = X_{i,i+|Y|-1}\,, \\ 0 & \textit{otherwise,} \end{cases} \\
\eta_X^{(k)}(Y,i,d) &= \eta_X^{(k-1)}(Y_{2,|Y|}) + \\
&\qquad \sum_{j>1:y_j=x_{i+1}} \sum_{l=0}^{k} \eta_X^{(l)}(Y_{2,j-1}) \eta_X^{(k-l)}(Y_{j,|Y|}, i+1, d) \cdot \\
&\qquad \sum_{s=0}^{l-1} \binom{l-1}{s} \binom{k-l-d+1}{s+1}.
\end{aligned}
$$

For completeness, we define $\eta_X^{(k)}(Y,i,d) = 0$ *for values* $d > k$.

This lemma is the analog to Lemma 6. Here, though, we cannot simply count the number of ways we can partition Y into k mutually nonoverlapping subsequences that correspond to substrings of X – we must consider how such a set of nonoverlapping subsequences might be ordered corresponding to a sequence of duplicate operations. Moreover, we must distinguish between generators that are comprised of the same set of substrings of X but that are ordered differently.

Intuitively, the value $\eta_X^{(k)}(Y)$ represents the number of length-k feasible generators for Y, the value $\eta_{Xi}^{(k)}(Y,i)$ represents the number of length-k f feasible generators for Y such that x_i generates y_1, and the value $\eta_X^{(k)}(Y,i,d)$ represents the number of length-k feasible generators for Y such that x_i generates y_1 and this character x_i appears in a substring of X that is d^{th} in the order of elements in the generator.

The recurrence given in Lemma 13 differs from that given in Lemma 6 in the inclusion of the function $\eta_X^{(k)}(Y,i,d)$. Fundamentally, the two additive terms in the definition of $\eta_X^{(k)}(Y,i,d)$ correspond to two cases that are analogous to the two cases originally described in the presentation of the duplication distance algorithm (see Thm. 1). In the first case, the substring corresponding to the character x_i generates the character at y_1 in a duplicate operation in which just a single character is copied, corresponding to an element of the generator. In this case, the remaining suffix $Y_{2,|Y|}$ is generated in another $k-1$ duplicate operations. For every length-$(k-1)$ feasible generator for the suffix $Y_{2,|Y|}$, we can insert

the substring x_1 into the d^{th} position in the ordering to yield a length-k feasible generator for Y in which the substring x_i generates y_1 and is ordered d^{th} in the generator. In the second case, the character at y_1 is generated in a duplicate operation along with another character at y_j (for some $j > 1$). In this case, the suffix of Y can be broken into two independent subproblems: the substring $Y_{2,j-1}$ and the suffix $Y_{j,|Y|}$. A length-l feasible generator for $Y_{2,j-1}$ and a length-$(k-l)$ feasible generator for $Y_{j,|Y|}$ can be combined to yield a length-k feasible generator for Y. Moreover, if the generator for $Y_{j,|Y|}$ contains a substring that begins at index $i+1$ in X to generate y_j and that substring is ordered d^{th} within that generator, then the combined length-k generator will have the necessary properties; in particular, the generator will include some substring that begins at x_i (and also includes x_{i+1} and possibly successive characters as well) that generates the characters y_1 (and y_j and possibly successive characters as well) and that substring will be ordered d^{th} in the generator.

Now, the last multiplicative term in the definition of $\eta_X^{(k)}(Y, i, d)$ accounts for the number of ways that we can construct a total order on k items that are partitioned into sets of l and $k-l$ items that are themselves, respectively, ordered. Note that the substring that generates the characters at y_1 and y_j must appear d^{th} in the combined total order and the substrings that comprise the length-l feasible generator for $Y_{2,j-1}$ (and therefore generate subsequences of Y that are inside the subsequence containing y_1 and y_j) must come after the d^{th} position in the ordering of all k elements. For an integer $0 \leq s \leq l-1$, a sequence of l items can be split at s positions in $\binom{l-1}{s}$ different ways, yielding $s+1$ subsequences. There are $k-l-d+1$ positions that come after the d^{th} position in between successive elements in the sequence of $k-l$ items comprising the feasible generator for $Y_{j,|Y|}$. There are $\binom{k-l-d+1}{s+1}$ ways of placing $s+1$ subsequences into these $k-l-d+1$ position to yield a totally ordered sequence of k items.

We can compute the restricted partition function $Q_X^{(k)}(Y)$ efficiently by first counting the number of relevant feasible generators, namely $\eta_X^{(k)}(Y)$, and scoring each generator appropriately by $\sigma(k, \mid Y \mid)$. This gives us the following theorem.

Theorem 11. *Let $X = x_1 \dots x_{|X|}$, $Y = y_1, \dots, y_{|Y|}$ be a source/target string pair and let k be a positive integer. The restricted partition function $Q_X^{(k)}(Y)$ satisfies the following.*

$$Q_X^{(k)}(Y) = \eta_X^{(k)}(Y) \cdot \sigma(k, \mid Y \mid).$$

To compute the recurrence in Lemma 13, we must compute the value $\eta_X^{(k)}(Y, i, d)$ for every substring of Y, every value in $\{i \mid x_i = y_1\}$, and every value $d = 1, \dots, k$; in total

$\eta_X^{(k)}(Y, i, d)$ must be computed $O(\mid Y \mid^2 \cdot \mu(X) \cdot k)$ times, where $\mu(X)$ is the maximum multiplicity of any character in X. Each computation of $\eta_X^{(k)}(Y, i, d)$ takes then $O(\mu(Y)k)$ time. Thus, the recurrence in Lemma 13 can be computed in time $O(\mid Y \mid^2 \mu(X)\mu(Y)k^2)$; the time to compute $Q_X^{(k)}(Y)$ is the same. In the worst case, this is $O(\mid Y \mid^5 \cdot \mid X \mid)$.

APPENDIX C: A DISCUSSION OF THE BLOCK ORDERING PROBLEM USING A BREAKPOINT GRAPH FRAMEWORK

The Block Ordering Problem was originally introduced by Gaul and Blanchette in [28]. The authors present a solution to the problem of completing a pair of partially ordered genomes so as to maximize the number of cycles in the resulting breakpoint graph. Recall from Section 4.3 that maximizing the number of cycles in the breakpoint graph for a pair of genomes is equivalent to maximizing the number of cycles in their adjacency graph. Therefore, the algorithm presented in Section 4.5.2 and the algorithm presented in [28] both produce a pair of complete genomes that satisfy the same optimality criterion.[1]

The solution in [28] begins with the construction of a *fragmented breakpoint graph*, a generalization of the breakpoint graph for a pair of genomes (see Section 4.3 for a description of a breakpoint graph). The nodes in a fragmented breakpoint graph for a pair of partial genomes on N genes correspond to the $2N$ extremities, and as in the breakpoint graph, the bi-colored edges between nodes correspond to adjacencies in either of the two genomes with each color corresponding to adjacencies in one of the two genomes. Obverse edges are omitted. However, because a partially assembled genome may exhibit fewer than N adjacencies, each extremity in the fragmented breakpoint graph may exhibit fewer than two neighbors. Thus, a fragmented breakpoint graph is comprised of a collection of color-alternating simple cycles and color-alternating simple paths. The process of completing the

[1] Note that the block ordering problem, as presented in [28], requires that the completed genomes be linear-unichromosomal. Although the algorithm in Section 4.5.2 constructs completed genomes that are circular-unichromosomal, the algorithm can be adapted easily to construct linear genomes instead by including a pair of odd-length paths in the final adjacency graph.

genomes optimally will result in the transformation of the collection of simple paths in the fragmented breakpoint graph into a set of simple cycles whose cardinality is maximum.

In order to complete the genomes optimally in [28], the authors construct a *block ordering graph* from the fragmented breakpoint graph. In this graph, the only vertices represented correspond to those with degree zero or one in the fragmented breakpoint graph (i.e. those for which adjacencies must be ascribed in order to complete the pair of genomes). Edges in the block ordering graph are either dashed or solid. Dashed edges connect pairs of vertices that appear at opposite ends of a simple path in the fragmented breakpoint graph, and solid edges connect pairs of vertices that appear at opposite ends of a block of adjacencies in one partial genomes.

The authors distinguish between different types of components in the block ordering graph and prescribe a method for "processing" each type of component. In particular, they identify components comprised entirely of solid edges corresponding to blocks from a single partial genome and dashed edges corresponding to paths in the fragmented breakpoint graph whose starting and ending vertices appear at the ends of blocks from the same partial genome, the so-called *one-sided components*. They note that the dashed edges can be "processed" by joining together the two endpoint vertices, creating a new adjacency in the genome and effectively merging together two blocks into one larger contig. Unfortunately, they note that "not all [such] edges can simultaneously be 'closed' that way, because this may lead to an invalid solution: since each [such] component is an alternating cycle in the [block ordering graph], closing all its dashed edges would correspond to joining all the corresponding block ends, ultimately resulting [in] a cycle of blocks." A cycle of blocks in a genome would correspond to a circular contig that cannot be merged with any other blocks by adding new adjacencies which is invalid. They then note that "the good news is that any [such dashed] edge can be sacrificed and the resulting partial ordering of blocks can be inserted anywhere in the complete orderings, without changing the score of the solution..." This relies on showing that for any pair of partial genomes whose block ordering graph exhibits a set of ω one-sided components with a total of l_α dashed edges among them, there cannot exist a linear-unichromosomal completion of the genomes that exhibit more than $l_\alpha - \omega$ new cycles derived from the processing of the dashed edges in one-sided components. However, this is not explicitly shown in [28].

In particular, if we suppose that a block ordering graph is comprised of ω one-sided components with a total of l_α dashed edges among them, then there must exist (by definition)

l_α alternating paths in the fragmented breakpoint graph. Therefore, a naive upper bound on the number of new cycles that can be constructed by closing those paths in the fragmented breakpoint graph is l_α. Instead, the strategy described in [28] produces $l_\alpha - \omega$ new cycles in the breakpoint graph by processing dashed edges. In the supplement to [28], the authors provide, in Lemma 2, that it is possible to construct a solution in which there are $l_\alpha - \omega$ new cycles in the fragmented breakpoint graph that result from processing one-sided dashed edges, but they do not show explicitly that $l_\alpha - \omega$ is an upper bound.

Note that in proving the optimality of our algorithm for the restricted block ordering problem in Section 4.5.2, we do prove an analog of the necessary lemma that is not explicitly shown in [28]. In Lemma 10, we state that the maximum number of cycles that can be added to a partial adjacency graph for a pair of partial genomes by completing the genomes is bounded by the number of missing adjacencies divided by two, $\frac{|\mathcal{M}(\mathcal{X})|}{2}$, (an analog of the number of dashed edges in the block ordering graph) minus the number of cycles in the genome graph defined by the genome obtained by augmenting the partial genome $\mathcal{X}$ with the set of unsatisfied pairs, $c(G_{\mathcal{X} \cup \mathcal{U}(\mathcal{A},\mathcal{X})})$ (the analog of the number of one-sided components in the block ordering graph). By then providing an algorithm that achieves this upper bound, we complete the proof of optimality stated in Thm. 10.